تعلَّم jQuery في 120 دقيقة

مُختار سيِّد صالح

تَعَلَّم
jQuery
في 120 دقيقة

مُختار سيِّد صالح

الإهداء

إلـى مَن يَرشَحُ حُبُّهُما من ثُـقـوبِ القَلبِ ،

.. إلـى والديَّ حفظهما الله .

المُقدِّمة :

شخصيّاً و إلى فترة قريبة - نوعاً ما - كنتُ أعتبرها من التقنيّات التي لا ينبغي لمطوِّر تطبيقات الـ **Web** أن يضيّع وقته في تعلُّمها و أراها من الأمور التي تدرّس في الكلّيات من باب الترّف العلمي ! ، نعم .. إنّها لغة **Java Script** التي تستخدم للبرمجة من طرف العميل ، كنتُ أقولُ لنفسي : " أنتَ تُبرمِجُ للـ **Web** بلغاتٍ عُليا و من طرف الخادم **Server** فلماذا تنظرُ للوراء ؟ " .

بعد فترةٍ ليست بالطّويلة أبدتْ ليَ الأيّامُ ما كنتُ جاهلاً و أتاني بالأخبارِ مَن لم أزوِّدِ فبدأتُ أدرك أهميّة البرمجة من طرف العميل و أغيّر قناعتي نادماً بعد ظهور تقنيّة **AJAX** التي فتحت أمام مبرمجي الـ **Web** حول العالم آفاقاً جديدةً للإبداع و بدأتُ أعودُ و أكتبُ بعضَ شيفرات الـ **Java Script** بغية تنفيذ ما يطلبه العملاء لكن و بينَ الحينِ و الحينِ كان موقفي السَّلبي من الـ **Java Script** يطفو على السَّطح لأنَّني كنتُ مضطرّاً لكتابة الكثير من شيفرات **Java Script** لتنفيذ أمور بسيطة - باعتقادي - و استمرَّ الحال هكذا إلى أن مَنَّ اللهُ عليَّ بمعرفة صديقة جديدة جميلة جداً اسمها **jQuery** الذي يلفظ "جَي كويري".

jQuery : مكتبة جديدة مجَّانيَّة و مفتوحة المصدر مكتوبة بلغة **Java Script** تسمح لكَ كمطوِّر لمواقع الـ **Web** بالقيام بما كان يتطلّب كتابة مئات الأسطر البرمجية بأسطر معدودة! و قد كتبها المبرمج الكبير **John Resig** في البداية (عام 2006) ثمَّ طوَّرها فريق من المبرمجين بالتعاون معه و الهدف من كتابتها تغيير الطريقة التي يكتب بها المبرمجون شيفرات الـ **Java script** على حدّ قول مبدعها.

نالتْ مكتبة **jQuery** خلال فترة قصيرة جداً شهرةً واسعةً أكسبتها ثقة مواقع أكبر الشّركات في العالم مثل **Google** و **Mozilla** و **WordPress** و **Drupal** و **DELL** و الكثير من المواقع الكُبرى التي كان آخرها موقع الشركة العملاقة **Microsoft** التي تبنّت

المكتبة و ضمّنتها بشكل افتراضي مع مشاريع لغة البرمجة ASP.NET 4 التي تُكتَب باستخدام بيئة Visual Studio 2010 و الجدير بالذكر أنَّ فكرة إنشاء مكتبات Java Script ليست فكرة جديدة فقد ظهر عدد كبير من المكتبات قبل ظهور مكتبة jQuery إلَّا أنَّ jQuery كانت الأكثر نجاحاً في مطابقة معايير الـ Web 2 و الأسهل على الإطلاق ، كما كانت مكتبة jQuery المكتبة الوحيدة التي ضمنت أعلى نسبة من التوافق مع جميع المتصفحات الشهيرة مثل IE6 فما يليه و Firefox 2 فما يليه و Safari 3 فما يليه و Opera 9 فما يليه و Google chrome و غيرها من المتصفحات و السبب الأهم في انتشار المكتبة و نجاحها هو صغر حجمها إذ أنّه لا يتجاوز الـ 20 كيلوبايتاً فقط !

حسناً .. كما يَعِدُ عنوان هذا الكتاب سنحاول تعلُّم هذه المكتبة معاً و إتقانها خلال ساعتين فقط! و الكلام هنا موجه لمن يعرف HTML و CSS و قليلاً من لغة Java script التقليدية حيث أنَّ المحتوى في هذا الكتاب – بإذن الله – ليس من ذاك الذي كثيراً ما نعاني منه في كتب المعلوماتية و خاصّةً ما يتحدث عن لغات البرمجة منها ، إذ نجد المؤلف يفجِّرُ في وجه القارئ عشرات الأسطر البرمجية غير المفهومة ثم ينتقل إلى فقرة أخرى قبل إزالة شظاياها عن وجه الالتباس ، و هو – إن شاء الله – ليس من الكتب التي لا تقول أي شيء عدا العناوين متعلِّلةً بعلل الاختصار الواهية ، و إنّما حاولتُ قدر استطاعتي جعل المحتوى في هذا الكتاب يقول ما يجب قوله فقط و اجتهدتُ كي تكون المعلومة فيه مكثفةً و بسيطةً و واضحةً بإذن الله ، ختاماً أسأل الله عزّ و جل أن يجعل عملي هذا خالصاً لوجهه و أن يكتبه من العلم الذي ينتفع به ، كما أدعوه جلّ جلاله أن تثبت التجربة لقارئ هذا الكتاب أنَّ عنوانه ليس عنواناً تجارياً و لا وعداً كاذباً تمرُّ به رياح الصَّيف دون نتائج ملموسة .

مُختار سيِّد صالح

من الدقيقة 0 إلى الدقيقة 8 :

أساسيَّات jQuery

تثبيت jQuery و تضمينها في صفحتك :

قبل أن تبدأ معي في الولوج إلى عالم المكتبة **jQuery** عبر دقائق هذا الكتاب يجب أن تقوم بتحميلها أولاً من خلال الدخول إلى موقعها الرسمي **www.jquery.com** و النقر على زر **Download** الواضح في الصُّورة :



بعد تحميل المكتبة بشكل صحيح يفترض أن يكون بحوزتك ملف بالاسم **jquery.js** و هي النسخة الكاملة من المكتبة و ملف آخر هو الملف **jquery-min.js** و هي نسخة خفيفة من المكتبة تمتلك كامل ميزات النسخة الكاملة مع اختلاف بسيط في الحجم إذ أنَّ النسخة الخفيفة ذات حجم أقل ، قم بنسخ أحد الملفين إلى المجلد الذي يحتوي مشروع الـ **Web** الخاص بك و أضف التعليمة التالية إلى رأس الصفحات التي ترغب باستعمال المكتبة فيها بين وسمَيْ **<head>** و **<head/>** :

```
<script src="jQuery.js" type="text/javascript">

</script>
```

و التعليمة السابقة هي تعليمة بسيطة – كما تعلم – تقوم بتضمين ملف **java script** مُعيَّن في صفحتك لتستخدم إمكانيّاته و وظائفه لاحقاً ، يتمُّ تحديد هذا الملف عبر الواصفة **src** التي تحمل قيمةً تعبِّر عن مسار هذا الملف بشكل مطلق أو بشكل نسبيّ و في السّطر السّابق فإن القيمة **jQuery.js** تعني مساراً نسبياً يشير إلى الملف ذو الاسم

jQuery.js الموجود في نفس المجلد الذي يحتوي مستند الـ Web الخاص بنا ، و هكذا نكون قد نجحنا في تضمين المكتبة ضمن الصفحة و بالتالي أصبحنا جاهزين للبدء في قراءة الدقائق التالية.

أساسيات jQuery :

تتكون مكتبة jQuery بشكل رئيسي من خمسة أقسام هي :

1- المُحَدِّدات Selectors. 2- الدَّوال Functions. 3- الأحداث Events.

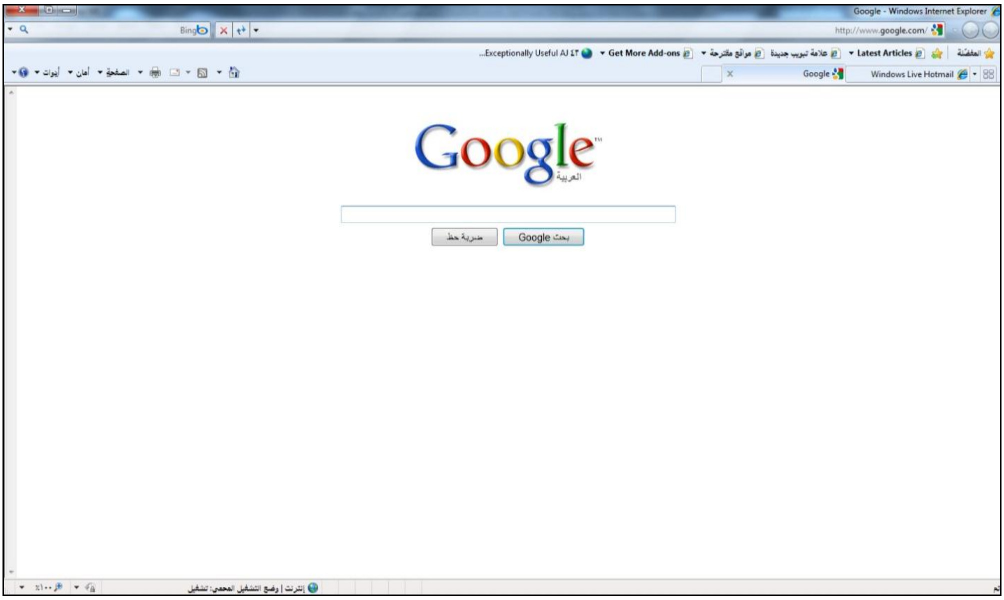
4- الحركات Animations . 5- الإضافات Plugins .

و كما قلتُ في المقدِّمة فإنَّ هذه المكتبة تسعى لتبسيط الأمور عند الحديث عن كتابة شيفرة Java Script لمنح تطبيقك مزيداً من التفاعلية إذ أنَّ المكتبة jQuery تقوم بتغليف مجموعة كبيرة من الأسطر البرمجية ضمن تابع بسيط جداً مما يسمح لك كمطور لتطبيقات الـ Web بالتركيز على وظيفة التطبيق فقط .

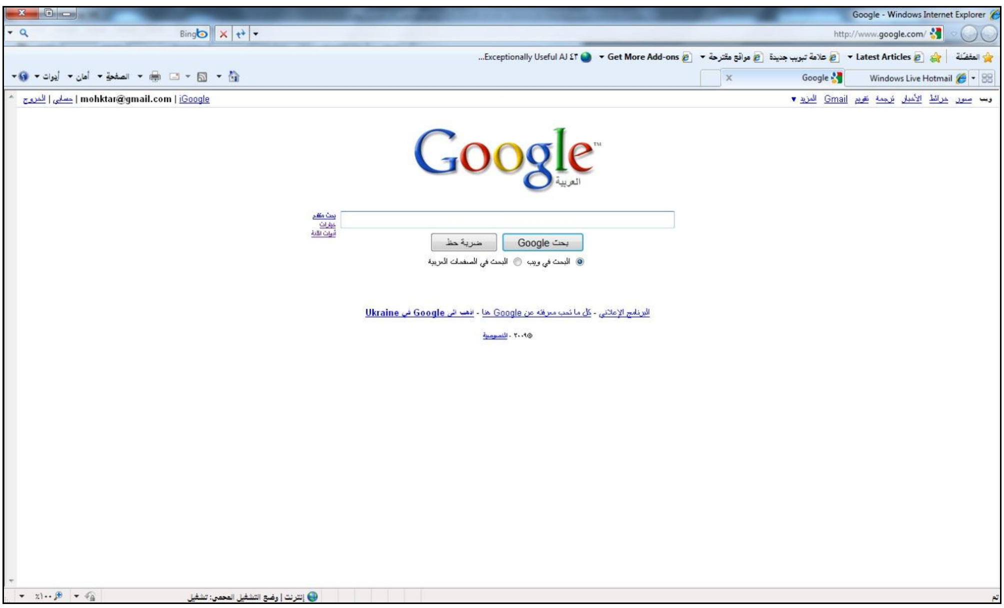
تتمحور الفكرة الرئيسية لمكتبة jQuery حول تطبيق تعديل أو حركة أو دالة ما على قسم محدد من صفحة الـ Web عند تفجير حدث معين ، و يتم تحديد القسم المراد إجراء التعديل عليه باستخدام المحدِّدات ، و بالطبع سنقسِّم الدقائق المتبقية لمناقشة كُلٍّ من أقسام المكتبة بشكل مفصّل و واضح بإذن الله.

و أنت يا صديقي الذي لم تتضح لك فكرة عمل المكتبة الرئيسية (يُفتَرضُ أنَّ أصدقائي كُثُر في هذه المرحلة) ما رأيك في مثال عملي يوضح أحد تطبيقات مكتبة jQuery على أرض الواقع ؟

ادخل للموقع الرائع Google لترى صفحة مشابهة لمايلي (حتى تاريخ كتابة هذا الكتاب) :



لاحظ أن الصفحة الخاصة بموقع محرك البحث Google لا تحتوي على أي شيء عدا ما هو ظاهر في الصورة (شعار الموقع و مربع البحث و الزرين الخاصين ببدء عملية البحث) ، و الآن حرّك مؤشر الفأرة في الصفحة و انظر ما سيحدث ! .. نعم سترى ما يشبه الصورة التالية:



لو انتبهت فإن ظهور المحتويات الجديدة في الصفحة (أعني الروابط **Links**) تم بحركة متلاشية جذابة تدعى بالظهور المتلاشي **Fade in** .

حسناً .. ما رأيناه هو مثال ممتاز يستخدم مكتبة **jQuery** فعلياً ، فهو مثال عملي على تطبيق <u>حركة</u> **Fade in** على <u>جزء من الصفحة</u> (الروابط **Links**) عندما تم <u>تفجير حدث معين</u> (مرور الفأرة في المنطقة الفارغة من الصفحة).

الأمر لحد الآن طبيعي لكن المذهل في الموضوع أن ما رأيته تم بكتابة سطر واحد من أبسط ما يكون و هو مشابه للسطر التالي :

```
$('a').fadeIn();
```

نعم كل هذا الجمال بسطر واحد فقط ! .

كمثال آخر ادخل لموقع شركة **Microsoft** الموضح في الصورة:



حاول أن تتجول في الصفحة و استمتع !

لا تدع هذا الجمال يخدعك لتعتقد أن الصفحة مبنية باستخدام برنامج **Flash** إذ أن معظم ما رأيته في الصفحة من قوائم **menus** و ألسنة **tabs** و حتى المزلاج **Slider** في نسخة سابقة منه يعتبر شيئاً بسيطاً من تطبيقات مكتبة **jQuery** الكثيرة (للأمانة : النسخة الحالية من المزلاج تستعمل تقنية **Silverlight**) .

المزيد من الأمثلة ؟

حسناً ..يمكنك الانتقال لدقائق استعراض أهم الإضافات في نهاية هذا الكتاب أو فعليك بموقع المكتبة نفسها مع أنني أرى ألَّا مثالَ أفضل من موقعَيْ **Google** و **Microsoft** أكبر شركتي برمجيّات في العالَم ، هذا العالَم ذاته الذي دفعته تجارب الشركات العملاقة للثقة بـ **jQuery** و هذا ما يدفعنا نحن أيضاً للبدء في الحديث عن طريقة استعمالها عسى أن نرى مواقعاً عربيةً تنافس بقوة في هذا المجال و هذا ليس بعيداً على وطنٍ عربيٍّ ينضحُ آلافَ المُبدعين الطَّموحين.

طريقة استعمال jQuery :

بشكلٍ عامٍّ و طاغٍ بعد أن تقوم بتضمين المكتبة ضمن الصفحة كما ذكرنا سابقاً ستقوم بكتابة شيفرة **Java script** عصريّة (و غير طويلة ☺) باستخدام مكتبة **jQuery** ليصبح شكل صفحتك النهائي كمايلي :

```
<html>
  <head>
     <script src="jQuery.js"  type="text/javascript">
```

```
    </script>

    <script type="text/javascript">

       $(document).ready(function() {

                    شيفرة جي كويري هنا

        });

    </script>

  </head>

  <body>

            محتوى صفحتك (الظاهر) سيكون هنا

  </body>

</html>
```

إن كان لك خبرة بسيطة جداً في كتابة مستندات HTML فأكاد أقسم أنك لم تواجه أية مشكلة مع ما سبق من شيفرة إلا مع الأسطر التالية :

```
$(document).ready(function() {

    شيفرة جي كويري هنا

});
```

و أكاد أقسم أيضاً أنك سترتاح إنْ قلت لك أنه ليس من الضروري أن تفهمها – الآن على الأقل– و لكن من المهم أن تتذكر شيئاً واحداً : من المحظورات البرمجية على أي مبرمج يعتمد على `jQuery` أن يكتب أي شيفرة خاصة بالمكتبة خارج المنطقة المخصصة لشيفرة

و هي الموضحة في الأسطر السابقة و من هذه النقطة إلى نهاية الكتاب بغية التسهيل و عدم الإطالة لن يذكر الكتاب كامل شيفرة الصفحة و إنما سيركز على الفقرة المطلوبة في كل مرة لكن أنت ستتذكر أن جميع شيفرات الـ jQuery توضع في هذه المنطقة.

و الآن يمكنك الانتقال مباشرةً للدقائق المخصصة للمحدِّدات و تجاهل الأسطر القليلة التالية إلا إنْ كنت ممّن يحب معرفة كل شيء فحينها يتوجَّب عليك أن تقرأ الفقرة التالية .

ما معنى الشيفرة السابقة ؟

تعتمد مكتبة jQuery على شجرة الـ DOM (شجرة كائنات المستند Data Object Model) بشكل كلي في عملها و كما تعلم فإن بناء هذه الشجرة لا يكتمل إلا بعد اكتمال تحميل الصفحة و في خطوط الانترنت البطيئة كالـ Dial up مثلاً قد يستغرق اكتمال تحميل الصفحة بعض الوقت الذي إن تم التلاعب بمكونات DOM أثناءه سيؤدي ذلك إلى إفساد شجرة كائنات المستند DOM و بالتالي إفساد صفحتك.

و لأنَّ مكتبة jQuery كُتِبَتْ في محاولة لجعل الحياة أسهل فإنَّ مهمة الأسطر السابقة ببساطة هي التحقق من اكتمال تحميل المستند و اكتمال بناء الشجرة قبل السماح بتنفيذ أي شيفرة jQuery حفاظاً على سلامة صفحتك و على سمعة المكتبة .

و الآن تستطيع القول بثقة أنك ممن يتقنون أساسيات مكتبة jQuery بالنسبة لمعايير هذا الكتاب .. مباركٌ مبارك.

من الدقيقة 8 إلى الدقيقة 32 :

المُحَدِّدات

Selectors

المُحَدِّدات :

هل تذكر كيف كان الحال قبل أوراق الأنماط Cascading Style Sheets أو ما يعرف اختصاراً بـ CSS ؟

كما تريد .. دعنا من المآسي و لننظر كم أصبحت حياة مصمم الـ Web جميلةً بعدها ، يكفي لتغيير حجم الخط الخاص بكافة النصوص المكتوبة ضمن الوسوم p أن تكتب في ورقة النمط:

```
p{ font-size:medium;  }
```

و يكفي لتغيير لون كافة الروابط الموجودة ضمن الصفحة أن تكتب في ورقة النمط :

```
a{ color:red; }
```

و يكفي لتغيير نوع الخط الخاص بكليهما معاً أن تكتب في ورقة النمط :

```
p,a{ font-family:Tahoma,Arial; }
```

و يكفي لتغيير عرض إطارات جميع الصور الموجودة ضمن وسمٍ ينتمي للصَّف Mukhtar أن تكتب في ورقة النمط :

```
.Mukhtar img{ border-width:medium; }
```

حسناً .. الكلام السابق ليس بعيداً عما أريد قوله خصوصاً إذا عرفت أنَّ استخدامنا لـ a و p و p,a و Mukhtar img. يدرسه متعلمو الـ CSS تحت عنوان المحدِّدات Selectors ، و كما يبدو من اسمها في jQuery فإن المحدِّدات تستعمل لتحديد مجموعة من عناصر مستند الـ Web (صفحة الـ HTML) في البداية ليتم تطبيق شيء من دوال أو حركات jQuery عليها لاحقاً.

هنالكَ عدد كبير من المحدِّدات التي تعطيك المرونة الكافية لتحديد ما تريده بالضبط و يكفي أن أخبرك أن محدِّدات CSS التي تعرفها ذاتها تعتبر بعضاً من محددات **jQuery** بما فيها محدِّدات النسخة CSS 3 (الـ CSS 3 لم تُطبَّق حتّى لحظة كتابة الكتاب و إنَّما صدرت معاييرها فقط) ممّا يعني أنك تعرف الكثير عن المحدِّدات مسبقاً.

لكي تقوم بتحديد عدد من عناصر الصفحة في **jQuery** يجب أن تكتب شيفرة **jQuery** بالشكل التالي :

```
jquery('selector');
```

أو :

```
$('selector');
```

حيث أن **Selector** يمثل المحدِّد الذي يعني مجموعة معينة من عناصر الصفحة و في هذه الدقائق سنتعرف على مختلف القيم التي يمكن أن تكون محددات صحيحة ، و في الحقيقة فإن **$** أو **jquery** يعتبران اسمين لتابع واحد و عليه فإن الجملتين **jquery(selector)** و **$(selector)** متكافئتان و لكنَّ غالبية المبرمجين يفضلون الشكل الثاني لأنه أقصر أو ربما لأنه يذكر البعض منهم بتعريف المتغيرات في PHP إن كانوا من مبرمجيها.

و كمثال عملي على الموضوع يمكن أن نقوم بتحديد جميع الروابط الموجودة في الصفحة باستخدام المحدد a الذي يعني جميع عناصر الوسم a الموجودة في الصفحة و عندئذٍ تكون التعليمة التي تقوم بهذه الوظيفة:

```
$('a');
```

أو :

```
jquery('a');
```

و يمكن أن نقوم بتحديد كافة الصور الموجودة في الصفحة باستخدام المحدد **img** بأحد الشكلين :

```
$('img');
```

أو

```
jquery('img');.
```

و كما تلاحظ فإن المحددين **a** و **img** هنا يستخدمان بنفس الصيغة الخاصة بمحددات أوراق الأنماط التي تعرفها ، و كمثال على استخدام محدد آخر يمكن أن نقوم بتحديد كافة خلايا الجداول الموجودة في الصفحة و التي تحتوي على صورة داخلها باستخدام المحدد **td:has(img)** بإحدى الصيغتين :

```
$('td:has(img)');
```

أو :

```
jquery('td:has(img)');
```

أتمنى أن تكون الفكرة الخاصة بالمحدِّدات قد اتضحت الآن لتقوم حينها بتجربة بعض شيفرات **jQuery** الخاصة بالمحدِّدات في صفحتك بشكل فعلي لتنتج صفحة مشابهة لمايلي:

```
<html>
  <head>
    <script src="jQuery.js" type="text/javascript">
    </script>
```

```
    <script type="text/javascript">

        $(document).ready(function() {

            $('a');

            $('img');

            $('td:has(img)');

        });

    </script>

  </head>

  <body>

        محتوى صفحتك (الظاهر) سيكون هنا

  </body>

</html>
```

قد تستغرب إن لم ترَ أية تغييرات على الصفحة بعد كتابة شيفرات **jQuery** الجديدة فيها و هنا يجب أن نوضّح أنَّ استخدام المحدِّدات لا يتم بمنأى عن دوال المكتبة حيث يجب أن يترافق استخدام المحدِّدات الخاصة بالمكتبة مع استخدام إحدى دوالها على الأقل ليَنتُجَ تغييرٌ ملموسٌ في الصفحة و هذا شيء سنتعلّمه بعد دقائق قليلة لكن حالياً يجدر بك أن تعرف أن استخدام الأسطر السابقة قام بتحديد ما تريده تماماً و لم يقم بأي تغيير و ما أريد قوله باختصار : "المُحدِّد يكتفي بتحديد جزء من الصفحة بينما تقوم الدوال بتغييره" ..هذه نقطة جوهرية يجب أن تفهمها قبل المتابعة .

ملحوظة : في نهاية الأمر $ و **jquery** ليسا إلا اسمين لتابع يعيد مجموعة العناصر التي يحددها المحدد **selector** على شكل مصفوفة من العناصر (النمط **array<element>** بالنسبة لمبرمجي **Java Script**) و هذا ما يعرف بالإنجليزية بـ **jQuery Wrapped Set** على رأي المكتبة و في توثيق المكتبة يدعونه أيضاً بالنمط **jQuery** اختصاراً .

حسناً .. الصيغة السابقة تكفي لتحديد جزء من عناصر المستند ، عندئذٍ يصبح بإمكانك تطبيق إحدى دوال (توابع) أو حركات المكتبة على الجزء المحدد بشكل مشابه لمايلي:

```
$('selector').function();
```

حيث أن **function** هو اسم الدالة أو اسم الحركة التي تريد تطبيقها ، و يعيد تنفيذ هذه الجملة (الدالة في الحقيقة) و غالبية جمل (دوال) **jQuery** مصفوفةً من العناصر من النمط **jQuery Wrapped Set** هي العناصر ذاتها التي قام المحدد **selector** بتحديدها و لكن بعد إحداث تغيير ما عليها ، و لنأخذ السطر التالي على سبيل المثال :

```
$('selector').hide();
```

يقوم المحدد **selector** بتحديد مجموعة من عناصر المستند تعيدها الدالة **$** كما هي بينما تقوم الدالة **hide** بإعادة هذه العناصر ذاتها بعد أن تغيِّر خاصية ظهورها إلى حالة عدم الظهور .

و من المهم أن نعرف أننا نستطيع تطبيق أكثر من دالة على المحتوى الذي يعيده أحد المحدِّدات فمثلاً يمكننا كتابة :

```
$('selector').function1().function2();
```

حيث أن السطر السابق سيقوم بتنفيذ الدالة **function1** على المحتويات التي يعيدها المحدد **selector** ثم يقوم بتنفيذ الدالة **function2** على المحتويات التي تعيدها الدالة **function1** و بما أن النمط الذي تعيده هذه الدوال هو نفس النمط الذي يمكن لها نفسها استقباله فهذا يعني أن الدالتين **function1** و **function2** ستطبقان على المحتوى الذي يعيده المحدد **selector** بنفس ترتيب استدعائهما و عليه فإن شيئاً كمايلي:

```
$('selector').Q1().Q2().Q3(). … .Qn();
```

يعتبر صحيحاً تماماً و هو شيء رائع لأن هذا يعني أنك تستطيع تنفيذ عدد **n** من الدوال على نفس العناصر التي يعيدها المحدد بحيث يكون خرج كل من هذه الدوال دخلاً للدالة التالية لها و كل هذه الروعة تعتبر تعليمةً واحدةً فقط لدى أهالي قبيلة **jQuery** !

فمثلاً يمكنك كتجربة واقعية أن تكتب مايلي :

```
$('a').fadeOut().addClass('red');
```

حيث ستقوم هذه الجملة عند تنفيذها بالبحث عن جميع الروابط في الصفحة (لأن المُحدِّد المستخدم **a** يعني جميع الروابط) ثم تقوم بتطبيق تأثير **fadeOut** عليها جميعاً و بعد ذلك تجعل قيمة الصف الخاص بورقة الأنماط **CSS Class** الذي تنتمي له هو الصف **red** (بمعنى آخر : تضيف الواصفة **class="red"** للعناصر المحددة) .

قبل أن نستعرض المحدِّدات كاملةً دعنا نكتب الشيفرة التالية لصفحة تستعمل **jQuery** بعد التأكد طبعاً من أن مسار **jQuery** المضمن و اسم ملف المكتبة صحيحان .

```
<html>
  <head>
    <title>أول اختبار لجي كويري الرائعة</title>
```

```
    <script src="jquery.js" type="text/javascript">
    </script>
    <script type="text/javascript">
      $(document).ready(function() {
        $('#Button1').click(function() {
          $('#TextArea1').toggle('slow');
        })
      });
    </script>
  </head>
  <body>
    <input id="Button1" type="button" value="انقرني لتجربة
المكتبة" />
    <textarea id="TextArea1" name="S1"></textarea>
  </body>
</html>
```

نتيجةً لتنفيذ الصفحة السابقة و النقر على الزر ستشاهد أبسط ما يمكنك عمله باستخدام مكتبة jQuery.

ما يهمني الآن أننا استعملنا الشيفرة التالية:

```
$('#TextArea1').toggle('slow');
```

للوصول إلى العنصر ذو المعرف TextArea1 في الصفحة و عمل ما شاهدته في المثال باستخدام الدالة toggle التي سنناقش عملها في دقائق الدوال و قد تم تحديد العنصر ذو المعرف TextArea1 عن طريق المحدد TextArea1# .

> ملحوظة : jQuery Wrapped Set ليست أكثر من مجموعة عناصر محددة ضمن صفحة الـ Web الخاصة بنا ، و سميت بهذا الاسم اصطلاحاً لأننا نقوم بتحديدها أولاً بغية القيام بنوع من عمليات jQuery عليها لاحقاً كالحركات أو الإضافات .

تذكّر : العناصر التي يعيدها استعمال الدالة $ أو jquery مع المحدِّدات تسمى jQuery Wrapped Set أو jQuery اختصاراً.

و لأن المحدِّدات الخاصة بالمكتبة كثيرة فإن تخصيص فقرة لكل منها سيؤدي إلى إنتاج كتاب من النوع الثقيل وزناً و الخفيف عِلماً ، و بدلاً من تخصيص فقرة مستقلة لكل محدِّد من المحدِّدات سنناقش جميع المحدِّدات في الجدول التالي بشكل يوضح عمل كل منها بسهولة و هذا أولاً و كي يكون لدينا مرجع سريع لجميع المحدِّدات (لمن يحب السرعة) ثانياً:

المحدد Selector	الوصف Description
*	يعني جميع العناصر الموجودة في المستند
T	يعني جميع عناصر الوسم T في المستند (مثلاً p يعني جميع عناصر الوسوم p التي في الصفحة و a يعني جميع عناصر الوسوم a و img يعني جميع عناصر الوسوم img و هكذا)
.C	يعني جميع عناصر الوسوم التي تنتمي لصف الأنماط C

	(الوسوم ذات الواصفة "class="C أياً كان نوعها)
.C_1 .C_2 .C_3 ... .C_n	يعني جميع الوسوم التي تنتمي لصفوف الأنماط المذكورة C_1 و C_2 و C_3 و C_n و كل ما تم ذكره
#X	يعني جميع عناصر الوسوم ذات المعرّف X (الوسوم ذات الواصفة "id="X أياً كان نوعها)
T F	يعني جميع عناصر الوسوم F الموجودة ضمن عناصر وسوم T بشكل مباشر و غير مباشر
T>F	يعني جميع عناصر الوسوم F الموجودة بشكل مباشر ضمن عناصر وسوم T (عندما نكتب مثلاً div>a فإننا نعني جميع الروابط الموجودة ضمن div بشكل مباشر فلو كان هذا الـ div ذاته يحتوي على جدول و أحد خلايا هذا الجدول تحتوي على رابط فلن يكون هذا الأخير من ضمن الروابط التي يعيدها المحدد ، عندئذٍ يمكننا أن نستخدم الصيغة السابقة div a لتحديد كل الروابط التي ضمن وسوم div بشكل مباشر أو غير مباشر)
T+F	يعني جميع عناصر الوسم F المجاورة لعناصر الوسم T (يقصد بالعناصر المجاورة العناصر الأبناء من نفس المستوى)

T:has(F)	يعني جميع عناصر الوسم T التي تحتوي داخلها عنصراً ابناً واحداً على الأقل من الوسم F (مثلاً Table:has(a) يعني أي جدول يحتوي رابطاً واحداً أو أكثر من رابط ضمن محتوياته)
T[A]	يعني جميع عناصر الوسم T التي تعرّف الواصفة A مهما كانت قيمة الواصفة A (مثلاً تستطيع أن تكتب المحدد a[href] لتعيد جميع الروابط التي تحتوي على واصفة href مهما كانت قيمتها)
T[A=value]	يعني جميع عناصر الوسم T التي تعرّف الواصفة A شرط أن تكون قيمة هذه الواصفة هي القيمة value (مثلاً تستطيع أن تكتب المحدد a[target=_blank] لتعيد جميع الروابط التي تفتح في نافذة هدف جديدة)
T[A^=value]	يعني جميع عناصر الوسم T التي تعرّف الواصفة A شرط أن تكون بداية القيمة التي تحتويها هذه الواصفة هي القيمة value (فمثلاً لو أردنا تحديد كافة الروابط التي تنقلك لموقع يبدأ بحرفي de فالمحدد a[href^=www.de] يلبي حاجتنا تماماً)
T[A$=value]	يعني جميع عناصر الوسم T التي تعرّف الواصفة A شرط أن تكون نهاية القيمة التي تحتويها هذه الواصفة هي القيمة value (المحدد a[href$=.mp3] على سبيل المثال يعني جميع الروابط التي تشير إلى ملفات

	من نمط mp3)
T[A*=value]	يعني جميع عناصر الوسم T التي تعرّف الواصفة A شرط أن تكون القيمة التي تحتويها هذه الواصفة تتضمن القيمة value (المحدد a[href*=great] يعني جميع الروابط التي يحتوي عنوانها على الكلمة great)
selector:first	يعني أول عنصر مطابق للمحدد selector على مستوى المستند (استخدام المحدد a:first على سبيل المثال يعيد أول عنصر رابط في المستند و استخدام المحدد div img:first يعيد أول عنصر صورة موجودة ضمن وسم div في المستند)
selector:last	يعني آخر عنصر مطابق للمحدد selector على مستوى المستند
selector:first-child	يعني أول عنصر ابن مطابق للمحدد selector على مستوى المحدد (إذا استخدمنا على سبيل المثال المحدد div a:first-child فإننا نعني كل أول رابط موجود ضمن كل وسم div في المستند و ليس أول حالة مطابقة للمحدد فقط كما هو الحال مع first)
selector:last-child	يعني آخر عنصر ابن مطابق للمحدد selector على

مستوى المحدد	
selector:only-child	يعني العنصر الابن المطابق للمحدد selector شرط أن يكون وحيداً في وسمه (لو أردنا أن نعيد جميع الروابط الموجودة ضمن وسوم div شرط ألا يوجد أكثر من رابط ابن مجاور في كل div فإن استخدام المحدد div a:only-child سيلبي مطلبنا)
selector:nth-child(n)	يعني العنصر الابن رقم n المطابق للمحدد selector (مثلاً لو أردنا تحديد الرابط الرابع الموجود في كل وسم div فإننا نكتب المحدد التالي div a:nth-child(4))
selector:nth-child(even)	يعني العناصر الأبناء ذات الأرقام الزوجية المطابقة للمحدد selector
selector:nth-child(odd)	يعني العناصر الأبناء ذات الأرقام الفردية المطابقة للمحدد selector
selector:nth-child(Xn)	يعني العناصر الأبناء ذات الأرقام التي من مضاعفات X (مثلاً في حالة كون الـ X يحمل القيمة 3 فإن المحدد النهائي يصبح div a:nth-child(3n) و يعني جميع الروابط الموجودة في وسوم div على أن يكون ترتيبها من مضاعفات العدد 3 و هذه الروابط هي الرابط الثالث و السادس و التاسع و الثاني عشر و الخامس عشر ...

	و هكذا)
selector:nth-child(Xn+Y)	يعني Y عنصراً ابناً بعد كل X أبناء (مثلاً في حال كون قيمة Y هي 2 و قيمة X هي 3 سيصبح شكل المحدد النهائي `div a:nth-child(3n+2)` و هذا يعني تحديد رابطين بعد كل ثلاثة روابط من أبناء div مما يعني الروابط الرابع و الخامس(رابطين بعد الرابط الثالث) و السابع و الثامن (رابطين بعد الرابط السادس) و العاشر و الحادي عشر (رابطين بعد الرابط التاسع) .... و هكذا)
selector:even	يعني العناصر ذات الأرقام الزوجية المطابقة للمحدد selector على مستوى المستند
selector:odd	يعني العناصر ذات الأرقام الفردية المطابقة للمحدد selector على مستوى المستند
selector:eq(n)	يعني العنصر ذو الترتيب n المطابق للمحدد selector على مستوى المستند
selector:qt(n)	يعني العنصر ذو الترتيب n المطابق للمحدد selector و جميع العناصر المطابقة التي تليه على مستوى المستند
selector:lt(n)	يعني العنصر ذو الترتيب n المطابق للمحدد selector و جميع العناصر المطابقة التي تسبقه على

	مستوى المستند
:animated	يعني جميع العناصر التي تخضع لحركة حالياً (سنناقش الحركات في دقائق قادمة)
:button	يعني جميع عناصر الأزرار الموجودة في المستند سواءً أكانت من النوع submit أو reset أو النوع button بشكل عام
:checkbox	يعني جميع عناصر صناديق الاختيار check boxes
:checked	يعني جميع عناصر صناديق الاختيار المحددة (أعني بالمحددة ما يكون حالتها checked سواء أكانت check box أو radio button)
:contains(s)	يعني جميع العناصر التي تحتوي على النص s (مثلاً يعيد المحدد p:contains(jQuery) جميع عناصر النصوص التي تحتوي على كلمة jQuery و يجدر الإشارة هنا أن هذا المحدد حساس لحالة الأحرف فالكلمة Yes تختلف عن yEs و تختلف عن YES)
:disabled	يعني جميع العناصر المعطلة (العناصر غير الفعالة)
:enabled	يعني جميع العناصر الفعالة
:file	يعني جميع عناصر اختيار الملفات و هو مكافئ للمحدد input[type="file"]

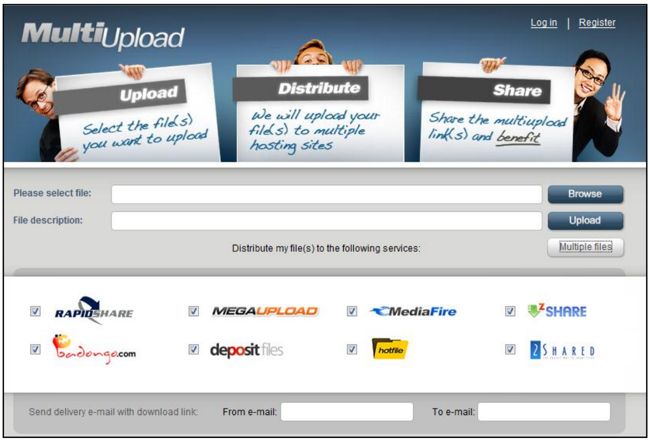
:header	يعني جميع عناصر الوسوم الخاصة بالعناوين h1,h2,h3,h4,h5,h6
:hidden	يعني جميع العناصر المخفية
:image	يعني جميع وسوم صور النماذج و هو مكافئ للمحدد input[type=image]
:input	يعني جميع وسوم النماذج input و select و textarea و button
:not(selector)	يعني جميع الوسوم التي لا تطابق المحدد selector (مثلاً يعيد المحدد :input:not(:button) جميع عناصر النماذج عدا الأزرار منها
:parent	يعني جميع عناصر الوسوم الآباء (التي تحتوي على عناصر وسوم فرعية بما فيها النصوص)
:password	يعني جميع عناصر إدخال كلمات المرور و هو مكافئ للمحدد input[type=password]
:radio	يعني جميع عناصر أزرار الاختيار radio button
:reset	يعني جميع أزرار إعادة التعيين
:selected	يعني جميع عناصر الاختيار المختارة (selected options)

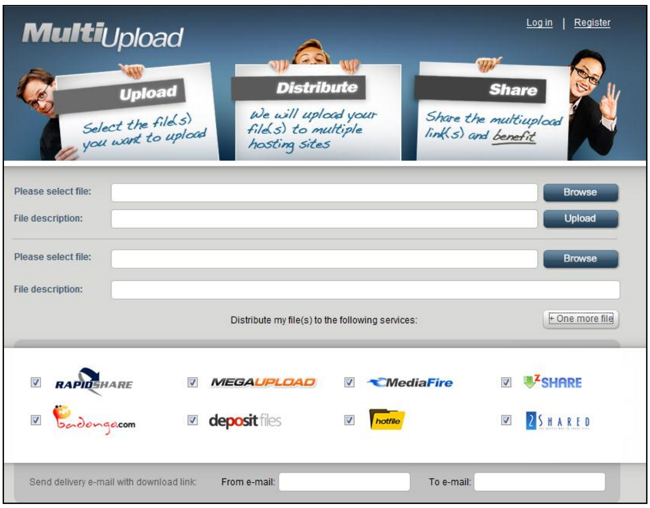
:submit	يعني جميع أزرار الإرسال
:text	يعني جميع مربعات إدخال النصوص و هو مكافئ للمحدد input[type=text]
:visible	يعني جميع العناصر الظاهرة
$Selector_1$,$Selector_2$,…,$Selector_n$	يعني اجتماع جميع عناصر الوسوم التي تعيدها المحدِّدات المذكورة و يقصد بالمحدد هنا كل ما هو مذكور في هذا الجدول و هذه القاعدة تعتبر مفيدةً جداً في حال الرغبة بتحديد أجزاء كثيرة من الصفحة لا يستطيع محدد واحد أن يقوم بتحديدها

لأنني أزعم أن هذا الكتاب هو أوَّل كتاب عربي يناقش مكتبة **jQuery** فيجدر بي التنويه إلى أن المكتبة لا تزال في إصدارها 1.3.2 حتى تاريخ تأليفه و هنالك المزيد من المحدِّدات الجديدة التي تضاف إلى كل إصدار جديد من إصدارات المكتبة الرائعة **jQuery** و يمكنك متابعة كل جديد عن المحدِّدات بشكل خاص و عن المكتبة بشكل عام عبر الدخول إلى موقعها الإلكتروني **www.jquery.com**.

توليد محتويات HTML جديدة :

في كثير من تطبيقات الـ Web الغنية Rich Internet Applications يتم توليد محتويات HTML جديدة إضافية للصفحة بعد عرضها فمثلاً في مركز رفع الملفات الشهير www.multiupload.com الظاهر في الصورة يسمح لك برفع ملف واحد كحد أقصى افتراضياً و في حال الحاجة لرفع المزيد من الملفات يمكنك بنقرة واحدة على الزر one more file أن تضيف ملفاً جديداً إلى مربعات تحديد الملفات في نفس الصفحة دون الحاجة إلى إعادة توليد الصفحة ، و لعلَّ الصُّورَ الجانبيَّة توضِّح الفكرة .





حيث يظهر في الصُّورة الثانية كيف تمت إضافة مربع جديد لرفع الملفات و لو نقرنا مرة أخرى على الزر one more file لأضيف مربع جديد آخر و هكذا ، و بالطبع لا داعيَ لإخبارك أن هذا يحدث في طرف العميل دون أي إعادة تحميل للصفحة و دون إرسال أي طلب للخادم Server ، هذا يعتبر مثالاً شائعاً جداً على إضافة محتويات HTML جديدة للصفحة و توفر المكتبة الرائعة jQuery هذه الإمكانية ببساطة

شديدة كما كل شيء فيها ، فيكفي لإضافة محتوى **HTML** جديد للصفحة أن تتبع الصيغة التالية:

```
$(HTML)
```

حيث أن **HTML** هنا تعني المحتوى الجديد الذي نريد توليده مكتوباً باستخدام لغة **HTML** العادية و كما تلاحظ فالصيغة هي ذاتها المستخدمة في المحدِّدات ، و إنَّ كتابة **$('<div>')** لإضافة عنصر **div** جديد للصفحة تكافئ كتابة **$('<div></div>')** كتسهيل للمبرمجين بمعنى أنه لا حاجة لإغلاق الوسم فالمكتبة تتولى ذلك نيابة عنا مع أنّني أفضِّل استخدام الشكل الكامل لمن يريد أن يطمئنَّ قلبه .

<u>ملحوظة :</u> يقولون : "لكل قاعدة شواذ" ، و كي لا تشذ المكتبة عن هذا القول فإنها للأسف الشديد لا تمكّننا من إنشاء عناصر الوسم script باستخدام الطريقة المذكورة.

عند استعمال هذه الصيغة فإن مكتبة **jQuery** تعيد المحتوى الجديد الذي تم إنشاؤه و لكنها لا تضيفه للصفحة لتعطيك بذلك حرية إضافته في المكان المطلوب تماماً من الصفحة فمثلاً يمكنك أن تكتب شيئاً مثل :

```
$('<div>محتوى جي كويري جديد</div>').append('#id');
```

حيث تقوم هذه التعليمة بإضافة المحتوى الجديد الذي ولدته في آخر المنطقة التي تمتلك المعرف **id** تماماً و هذه هي وظيفة الدالة **append** إحدى دوال المكتبة المخصصة لهذا الموضوع و التي سنتعرّف على عملها و عمل أخواتها بالتفصيل في الدقائق التالية .

من الدقيقة 32 إلى الدقيقة 57 :

الدَّوال

Functions

الدَّوال Functions:

و الآن .. بعد تطبيق أحد المحدِّدات باستخدام الطريقة التي تعلّمناها قبل قليل و إعادته لمجموعة من عناصر صفحتنا أو بعد توليد هذه العناصر فإنّنا نصبح جاهزين لتطبيق شيء من دوال jQuery أو حركاتها على هذا المحتوى (jQuery Wrapped Set) لنعطي الحياة لصفحتنا و هذا ما سنناقشه في هذه الدقائق ، و سنقوم تسهيلاً بتقسيم هذه الدوال إلى مجموعات حسب الوظيفة الخاصّة بكلٍّ منها.

دوال التعامل مع واصفات الوسوم Attributes :

لو نظرنا للوسم التالي على سبيل المثال :

```
<img src="images/pic1.gif" alt="imageOne" title="image one" width="40" height="20">
```

لعرفنا مباشرةً أنّها صورة تمتلك واصفاتٍ معيّنة Attributes مثل الارتفاع height و العرض width و العنوان title و النّص البديل alt و مصدر الصورة src و هذا ما يمثله مستعرض الـ Web بعقدة node من عقد شجرة كائنات المستند ،لهذه العقدة مجموعة من الخصائص Properties تقابل كل خاصية منها واصفةً من واصفات الوسم (بلى يا صديقي المبرمج .. عقد شجرة المستند تشبه الكائنات في لغات البرمجة غرضية التوجه) .

<u>ملحوظة :</u> بعد إنشاء شجرة كائنات المستند فإن أي تغيير في خصائص إحدى العقد سينعكس مباشرةً على قيمة الواصفة المقابلة للوسم المقابل في الصفحة و هذا تفسير آخر لمنع المكتبة إيّانا من تنفيذ أوامرها قبل اكتمال بناء هذه الشجرة.

تسمح لنا مكتبة jQuery بالتعامل مع هذه الواصفات أو الخصائص حقيقةً باستخدام طريقة بسيطة جداً فيكفي لاستعادة قيمة إحدى هذه الواصفات أن نستعمل الدالّة attr بالشكل التالي:

```
$('#id').attr('attributeName');
```

لنستعيد قيمة الواصفة attributeName الخاصة بالوسم ذو المعرف id و يكفي لإسناد قيمة لإحدى هذه الواصفات أن نستعمل الدالّة attr بالشكل التالي:

```
$('#id').attr('attributeName' , 'value');
```

لنجعل قيمة الواصفة attributeName الخاصة بالوسم ذو المعرف id مساويةً للقيمة value.

أعتقد أنك لاحظت أنني استعملت المحدد id# الذي يعيد عنصراً واحداً في العادة (إذا لم تضع معرفات مكرّرةً في صفحتك) ثم قمت بتغيير إحدى واصفات هذا العنصر و السّؤال الذي يطرح نفسه الآن هل ستنجح هذه التعليمة في حال التعامل مع محدّدات تعيد مجموعة من العناصر ؟

توفّر المكتبة جواباً لهذا السؤال عبر تأمين طريقة للتعامل مع واصفة ما لمجموعة من العناصر عن طريق الدوران على جميع تلك العناصر ، فيكفي لقراءة واصفة معينة لجميع عناصر مجموعة معينة أن تستخدم الصيغة التالية :

```
$('selector').each(function(n){
  $(this).attr('attributeName');
});
```

و يكفي لإسناد قيمة واصفة معينة لجميع عناصر مجموعة معينة أن تستخدم الصيغة التالية :

```
$('selector').each(function(n){
  $(this).attr('attributeName' , 'value');
});
```

تذكّر : في هذه الحالة فإنّ الكلمة المحجوزة **this** تشير إلى العنصر الحالي أثناء الدوران **each**.

و لمن لا يحب الدوران **each** و لا يريد أن يعود ليتذكر الحلقات التكرارية توفر المكتبة طريقة أخرى أسهل لإسناد قيمة واصفة أو مجموعة من قيم الواصفات لجميع عناصر المجموعة التي يعيدها المحدد عن طريق استخدام الصيغة التالية:

```
$('selector').attr(JSON);
```

حيث أن **JSON** هو كائن يمثل تلك الواصفات و من لا يعرف **JSON** له أن ينتقل إلى الملحوظة التالية ثم يعود أو بإمكانه أن ينظر معي للمثال التالي لعلّه يكتشف أنها مجرد طريقة لتمثيل عناصر (عقد) شجرة المستند بشكل مستقل عن النظام :

```
$(':input').attr(
     { value:'' , title:'يرجى إدخال قيمة في هذا الحقل' }
);
```

و كما تتوقع فإن الأسطر أعلاه تقوم بإسناد القيمة النصية الفارغة '' لخاصية **value** الخاصة بجميع مربعات النصوص في الصفحة و تقوم بإسناد القيمة '*يرجى إدخال قيمة في هذا الحقل*' للخاصية **title** الخاصة بجميع مربعات النصوص في الصفحة .

و الآن أصبحنا نعرف كيف نقوم "بالقراءة من" و "الكتابة إلى" واصفات العناصر و يبقى أن نعرف أنه يمكننا حذف الواصفات أيضاً باستخدام الدالّة removeAttr التي لها الشكل التالي :

```
$('#id').removeAttr('attributeName');
```

حيث أنَّ attributeName يمثِّل اسم الواصفة التي نريد حذفها و كمثال على استخدام الدالة removeAttr قد نكتب مايلي:

```
$('a').removeAttr('target');
```

لحذف الواصفة target من جميع الروابط في المستند .

ملحوظة : **JSON هي اختصار لـ Java Script Object Notation أي ترميز كائن جافا سكربت و هذا الترميز بسيط جداً جداً على عكس اسمه المخيف إذ أنه يقوم بتمثيل خصائص الكائن بشكل أزواج (مفتاح / قيمة) على الشكل التالي :**

```
{
      Key1 : 'value1' ,
      Key2 : 'value2' ,
      … ,
      Keyn : 'valuen'
}
```

حيث أن المفتاح (الخاصية) key_1 يحمل القيمة $value_1$ و المفتاح key_2 يحمل القيمة $value_2$ و المفتاح key_n يحمل القيمة $value_n$ و هكذا .

دوال التعامل مع الأنماط Styles :

تؤمّن لنا مكتبة jQuery القدرة على التعامل مع أنماط (ستايلات) كائنات الصفحة عن طريق مجموعة من الدّوال ، حيث تسمح لنا بإضافة نمط جديد (CSS Class جديد) للعناصر عن طريق الدالة addClass التي تستعمل كمايلي:

```
$('selector').addClass('className');
```

حيث تقوم هذه الدالة بإضافة النمط className المعرّف ضمن ورقة الأنماط الخاصة بالصفحة إلى مجموعة العناصر التي يعيدها المحدد selector و هنا من الضروري أن نتذكر أن أي عنصر في المستند يستطيع أن ينتمي لأكثر من صف في ورقة الأنماط شرط أن نفصل بين أسماء الصفوف بمحرف المسافة space و لذا لا نستغرب عندما نرى شيئاً مماثلاً لما يلي :

```
<span class="class1 class2 class3 class4 … classn"> </span>
```

و لا نستغرب عندما يتم تطبيق كافة خصائص الصفوف المذكورة على العنصر span و هذا سبب تسمية الدالة التي ناقشها بـ addClass لأن عملها الفعلي هو إضافة نمط جديد إلى العناصر التي يعيدها المحدد بشكل إضافي للأنماط الموجودة أصلاً.

على النقيض تماماً تسمح لنا مكتبة jQuery بحذف أحد صفوف الأنماط التي ينتمي لها كائن معين عن طريق الدالّة removeClass التي تستخدم بالشكل :

```
$('selector').removeClass('className');
```

و من الطبيعي أن تقوم هذه الدالة بإيقاف تطبيق خصائص النمط **className** على مجموعة العناصر التي يعيدها المحدد **selector** .

من الدوال الجميلة جداً التي تؤمنها المكتبة **jQuery** أيضاً هي دالة القلب **toggleClass** التي تقوم بإضافة أحد صفوف النمط إلى العناصر التي لا تنتمي إليه من المجموعة التي يعيدها المحدد و تقوم بنفس الوقت بإيقاف تطبيق ذات الصف على العناصر التي تنتمي إليه من المجموعة التي يعيدها المحدد و لهذه الدالة الشكل التالي :

```
$('selector').toggleClass('className');
```

و قد يبدو عمل هذه الدالة مستهجناً في الوهلة الأولى و لكن إن نظرنا إلى المثال التالي سنرى الفائدة الكبيرة التي تقدمها هذه الدالة ، لنفترض جدلاً أننا قمنا بكتابة الصفحة التالية:

```
<html>
<head>
  <title>قبل اختبار دالة القلب</title>
  <style type"=text/css">
        .black {
          background-color:black;
           color:White;
        }
  </style>
</head>
<body>
```

```
<table id="table1">

  <tr>

    <td>اسم المنتج</td>

    <td>سعر المبيع</td>

  </tr>

  <tr>

    <td>منتج 1 </td>

    <td>سعر 1</td>

  </tr>

  <tr>

    <td>منتج 2</td>

    <td>سعر 2</td>

  </tr>

  <tr>

    <td>منتج 3</td>

    <td>سعر 3</td>

  </tr>

  <tr>

    <td>منتج 4</td>
```

```
    <td>سعر 4</td>

   </tr>

  </table>

</body>

</html>
```

و هي صفحة بسيطة جداً تحتوي جدولاً لأسماء و أسعار منتجات وهمية و نتيجة عرضها على الشاشة تشبه ما يلي :

سعر المبيع	اسم المنتج
سعر1	منتج1
سعر2	منتج2
سعر3	منتج3
سعر4	منتج4

بعد فترة و أثناء تجولنا على الشبكة العالمية **Internet** رأينا أن معظم مواقع الـ **Web** الجيدة تقوم بعرض أسطر الجداول بلونين متناوبين و هو ما يعرف بالإنجليزية بـ (**Zebra stripping**) بالشكل :

سعر المبيع	اسم المنتج
سعر 1	منتج 1
سعر 2	منتج 2
سعر 3	منتج 3
سعر 4	منتج 4

فتحمسنا لجعل جدولنا يبدو كبقية الجداول الجيدة في عالم الـ Web و قمنا بتعديل شيفرة الجدول في صفحتنا لتصبح كمايلي :

```
<table id="table1">
  <tr>
    <td>اسم المنتج</td>
    <td>سعر المبيع</td>
  </tr>
  <tr class="black">
    <td>منتج 1</td>
    <td>سعر 1</td>
  </tr>
  <tr>
    <td>منتج 2</td>
    <td>سعر 2</td>
  </tr>
  <tr class="black">
    <td>منتج 3</td>
    <td>سعر 3</td>
```

```
  </tr>

  <tr>

    <td>منتج 4</td>

    <td>سعر 4</td>

  </tr>

</table>
```

و بعد نجاح تعديلنا بفترة لنفترض أننا أردنا تغيير ترتيب تناوب اللونين لسبب ما لنجعل شكل الجدول كما يلي :

اسم المنتج	سعر المبيع
منتج 1	سعر 1
منتج 2	سعر 2
منتج 3	سعر 3
منتج 4	سعر 4

قد نخطئ هنا ذات الخطأ السابق و نعود لتعديل شيفرة الجدول في حين أن كتابة السطر التالي من أسطر المكتبة jQuery ستحل المشكلة :

```
$('#table1 tr').toggleClass('black');
```

و هنا لا بد من الاعتراف أن السطر التالي كان سيجعل الحياة أسهل في الحالة السابقة عندما أردنا تلوين بعض سطور الجدول :

```
$('#table1 tr:even').addClass('black');
```

إلى الآن تعاملنا مع أنماط العناصر عبر الصفوف Classes و لكن ماذا لو أردنا أن نقوم بالتعامل مع كل خاصية من خصائص أنماط العنصر بشكل مباشر دون الحاجة لصف ؟

توفر المكتبة ذلك أيضاً عبر الدالة css التي تسمح لنا بقراءة قيمة خاصية نمط العنصر بشكل مباشر عبر الصيغة التالية:

```
$('#id').css('name');
```

حيث ستقوم الدالة بإعادة قيمة الخاصية name من خصائص أنماط العنصر الذي يحمل المعرف id ، أما لإسناد قيمة خاصية نمط لعنصر معين يمكن استعمال الصيغة :

```
$('#id').css('name','value');
```

التي ستقوم بإسناد القيمة value إلى الخاصية name من خصائص أنماط العنصر الذي يحمل المعرف id ، و كما قلنا سابقاً عند مناقشة الدالة attr فإننا نستطيع عمل دوران each "للقراءة من" أو "للكتابة إلى" أكثر من عنصر أو يمكننا الاستغناء عن دوران each باستعمال الصيغة المختصرة :

```
$('#id').css(JSON);
```

حيث يتم تمرير أزواج (مفتاح / قيمة) في كائن JSON تعبر عن مجموعة من خصائص أنماط العنصر ليتم إسنادها دفعةً واحدةً للعناصر التي يعيدها المحدد selector .

هناك دوال أخرى للتعامل مع ما يعتبر من خصائص أنماط العنصر مثل الدالة width التي تسمح بقراءة عرض العنصر بالشكل :

```
$('#id').width();
```

أو إسناد عرض العنصر بالشكل :

```
$('#id').width(value);
```

و مثلها الدالة `height` التي تسمح بقراءة ارتفاع العنصر بالشكل :

```
$('#id').height();
```

أو إسناد ارتفاعه بالشكل :

```
$('#id').height(value);
```

دوال التعامل مع محتوى عناصر المستند Inner content :

تعطينا المكتبة `jQuery` إمكانيات أخرى للتعامل مع مستند الـ `Web` و هذه المرة تسمح لنا بالتعامل مع محتوى العناصر عبر مجموعة من الدوال منها دالة `html` التي تقوم بقراءة محتوى عنصر معين عبر الشكل :

```
$('#id').html();
```

حيث أنها تعيد شيفرة الـ HTML التي تمثل المحتوى الموجود داخل العنصر ذو المعرف `id` و تقوم بإسناد قيمة جديدة للمحتوى(استبداله) بالشكل :

```
$('#id').html(Con);
```

حيث أن الوسيط `Con` هو شيفرة الـ HTML التي تمثل المحتوى الجديد.

أما الدالة `text` فهي مماثلة للدالة السابقة عدا أنها تتعامل مع المحتوى كنص عادي و ليس كشيفرة HTML و لها الشكل التالي في حالة القراءة :

```
$('#id').text();
```

و الشكل التالي في حالة الإسناد:

```
$('#id').text(Con);
```

حيث أن الوسيط **Con** هو نص عادي يمثل المحتوى الجديد.

و في سبيل المقارنة بين الدالة **html** و الدالة **text** لو كان لدينا في صفحتنا ما يلي مثلاً :

```
<ul id="myul">
     <li>1</li>
     <li>2</li>
</ul>
```

ثم قمنا باستدعاء الدالة **text** كما يلي :

```
$('#myul').text();
```

فإنها ستعيد القيمة **12** كنص (**String**) أما في حالة استدعاء الدالة **html** بنفس الصيغة فإنها ستعيد المحتوى :

```
<li>1</li>
<li>2</li>
```

لا يقتصر الموضوع على الاستبدال الكلي لمحتوى وسوم المستند و إنما يمكن تعديل جزء من محتوى المستند و لعلَّ الدالّة **append** خير ما نستهلُّ به حديثنا عن هذه النقطة فهي تقوم بإضافة محتوى **HTML** جديد إلى نهاية العناصر المحددة و تستعمل بالشكل :

```
$('selector').append(HTML);
```

و كنا قد رأينا مثالاً على عملها هذا في فقرة سابقة حيث استخدمناها لتوليد محتوى جديد و إضافته للصفحة و لا داعيَ لإعادة الكلام هنا و لكن ما أريد قوله في هذه الفقرة أن لهذه الدالة عملاً آخراً هاماً جدّاً يمكّننا من نقل أو نسخ مجموعة من عناصر المستند من مكانها ضمن المستند إلى مكان آخر ضمنه فمثلاً يمكن أن نكتب مايلي :

```
$('targetSelector').append($('sourceSelector'));
```

و هنا تلاحظ أننا قمنا بتمرير مجموعة من العناصر للدالة عن طريق استدعاء محدد sourceSelector و لم نقم بتمرير محتوى HTML بشكل مباشر كما فعلنا في المثال الذي ناقشناه سابقاً و في هذه الحالة ستقوم هذه التعليمة بأخذ العناصر التي يعيدها المعرف sourceSelector و نقلها من مكانها الأصلي إلى المكان الهدف مما يعني حذفها من المكان الأصلي و إضافتها في المكان الهدف و هو آخر العنصر الذي يعيده المحدد targetSelector إن كان ما يعيده هو عنصراً واحداً أما إن كان أكثر من عنصر فإن الدالة ستقوم بعملية نسخ بدل النقل مما يعني أنها ستحافظ على المحتوى المصدر في مكانه و تضيف محتوى مماثل له تماماً آخر كل كائن يعيده المحتوى الهدف و كمثال على الموضوع تصور أن لدينا المحتوى التالي في المستند:

```
<ul>

    <li>11</li>

    <li id="item12">12<li>

</ul>

<ol>

    <li>21<li>

</ol>
```

في مثل هذه الحالة فإن الاستدعاء :

```
$('#item12').append($('ol'));
```

سيقوم بحذف القائمة الثانية من موضعها الأصلي و إضافة واحدة جديدة مطابقة لها تماماً و جعلها جزءاً من القائمة الأولى تابعاً للعنصر الثاني في نهايته في حين أن الاستدعاء :

```
$('ul li').append($('ol'));
```

سيقوم بنسخ القائمة الثانية دون أن يأثر عليها و ينشئ قائمة مطابقة لها تماماً يضيفها إلى نهاية كل عنصر من عناصر القائمة الأولى .

تذكّر : إذا كان المحدد الهدف يعيد أكثر من عنصر (مصفوفة من العناصر) فإن استدعاء الدالة **append** يقوم بنسخ المصدر أما إنْ كان ما يعيده المحدد الهدف عنصراً واحداً فإن استدعاء الدالة يقوم بنقل المصدر .

هناك دالة مشابهة بالاسم فقط للدالة **append** اسمها **appendTo** تستدعى بالشكل التالي:

```
$('source').appendTo('target');
```

حيث تقوم هذه الدالة بنقل العناصر التي يعيدها المحدد **source** و إضافتها إلى آخر العنصر الذي يعيده المحدد **target** إن كان يعيد عنصراً واحداً و إن كان يعيد أكثر من عنصر فإنها تقوم بنسخ هذه العناصر و ليس نقلها و على سبيل المثال فإن التعليمة :

```
$('#copyrightLabel').appendTo('.code');
```

تقوم بنقل العنصر ذو المعرف **copyrightLabel** إلى العنصر الذي ينتمي لصف النمط **code** في حال كان وحيداً و إن كان هناك أكثر من عنصر ينتمي للصف **code** في الصفحة سيتم نسخ العنصر ذو المعرف **copyrightLabel** و ليس نقله .

<u>ملحوظة :</u> لو انتبهت لطريقة عمل الدالة **append** من حيث البارمترات و قارنتها بالدالة **appendTo** لوجدت أنهما متعاكستان من ناحية كون أي البارمترات مصدراً و أيها هدفاً .

على عكس الدالتين **append** و **appendTo** اللتين تقومان بإضافة المحتوى في نهاية المحتوى الداخلي للعناصر المحددة (قبل وسم الإغلاق الخاص بكل منها) فإن الدالتين **prepend** و **prependTo** تعملان بنفس الطريقة تماماً عدا أنهما تضيفان المحتوى الجديد في بداية المحتوى الداخلي للعناصر المحددة (بعد وسم البدء لكل منها) حيث تعمل **prepend** بنفس طريقة **append** و تعمل **prependTo** بنفس طريقة **appendTo** .

أما بالنسبة للدالتين **before()** و **insertBefore()** فإنهما تعملان بأسلوب مشابه أيضاً مع ملاحظة أن الدوال السابقة تضيف المحتوى في بداية أو نهاية المحتوى الداخلي للهدف (بعد وسم البدء أو قبل وسم الإغلاق) في حين أن هاتين الدالتين تقومان بإضافة المحتوى قبل الهدف تماماً (قبل وسم البدء الخاص به) .

و بالمثل توفر المكتبة دالتين تعملان بنفس الأسلوب مع اختلاف بسيط إذ أنهما تضيفان المحتوى بعد الهدف تماماً (بعد وسم الإغلاق الخاص به) و هما **after()** و **insertAfter()** .

و لأن الدوال الجديدة التي قرأتها قبل قليل تعمل بطريقة مشابهة للدالة **append** سأكتفي بذكر مثال واحد مختصر فقط على كل منها .

تقوم التعليمة التالية :

```
$('p img').before('<p>I love Syria!</p>');
```

بإضافة الجملة **I love Syria** قبل كل عنصر يعيده المحدد **p img** ، و تقوم التعليمة التالية بنفس المهمة :

```
$('<p>I love Syria!</p>').insertBefore('p img');
```

بينما تقوم التعليمة التالية :

```
$('a[href^=http://www.]').after('<small style = "color:red">
external </small>');
```

بإضافة الكلمة **external** بعد كل رابط يشير إلى موقع خارجي ، و تقوم التعليمة التالية بالوظيفة ذاتها :

```
$('<small style="color:red"> external
</small>').insertAfter(' a[href^=http://www.]');
```

كل دالّة من الدوال الجميلة السابقة لها عمل معين قد يشابه غيرها و لكن اختلاف الأسلوب يجعل كلاً منها ملائمة تماماً لحالات معينة و الجدول التالي يساعد على تلخيص عملها :

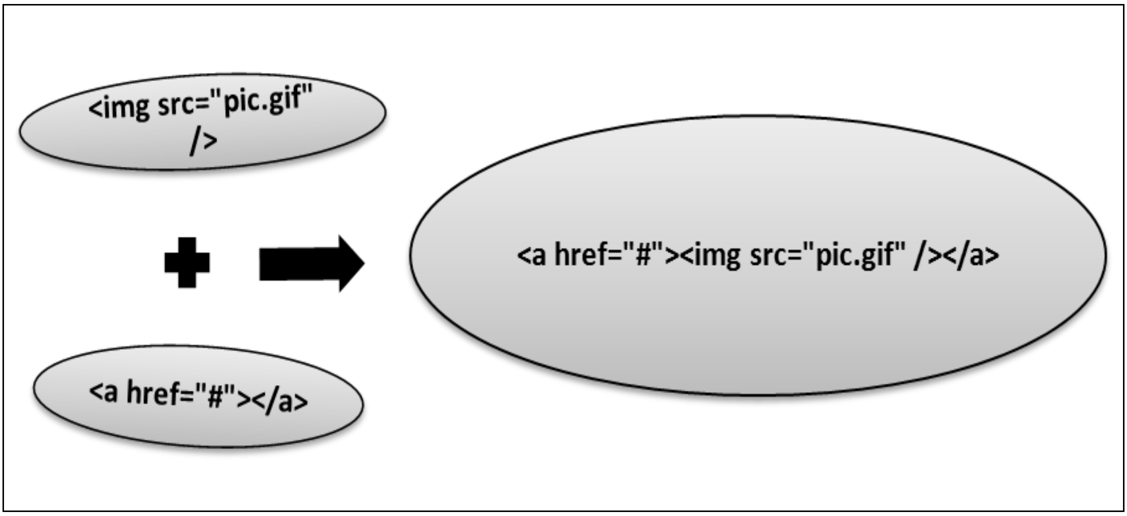
اسم الدالة	الوظيفة
html()	في حال استدعائها بدون تمرير أي وسيط **Parameter**

	تقوم الدالة بإعادة محتوى العناصر المحددة على شكل شيفرة HTML ، و في حال تمرير وسيط مناسب لها تستبدل المحتوى السابق للعناصر المحددة به .
text()	تشبه بعملها عمل html مع فرق أن ما تأخذه كوسيط في حال تمرير وسيط لها و ما تعيده في حال عدم تمرير وسيط لها هو نص String .
append()	تضيف محتوى الوسيط الممرر لها إلى نهاية المحتوى الداخلي لكل من العناصر التي يعيدها المحدد كما تستعمل في حال الرغبة في نقل / نسخ مجموعة من عناصر المستند من مكان ضمن المستند إلى مكان آخر ضمنه .
appendTo()	تضيف المحتوى الذي تطبق عليه إلى نهاية المحتوى الداخلي لكل من العناصر التي يعيدها وسيطها الذي يكون محدداً .
prepend()	تضيف محتوى الوسيط الممرر لها إلى بداية المحتوى الداخلي لكل من العناصر التي يعيدها المحدد .
prependTo()	تضيف المحتوى الذي تطبق عليه إلى بداية المحتوى الداخلي لكل من العناصر التي يعيدها وسيطها الذي يكون محدداً .
before()	تضيف محتوى الوسيط الممرر لها قبل كل عنصر من العناصر التي يعيدها المحدد.

insertBefore()	تضيف المحتوى الذي تطبق عليه قبل كل عنصر من العناصر التي يعيدها وسيطها الذي يكون محدداً .
after()	تضيف محتوى الوسيط الممرر لها بعد كل عنصر من العناصر التي يعيدها المحدد.
insertAfter()	تضيف المحتوى الذي تطبق عليه بعد كل عنصر من العناصر التي يعيدها وسيطها الذي يكون محدداً .

دوال التغليف Wrapping :

أحياناً نحتاج للقيام بتغليف محتوى ما بوسم معيّن أو مجموعة من الوسوم فمثلاً لنفترض أننا نريد تغليف جميع وسوم img بالوسم a مثلاً (جعل وسوم img محتويات لوسوم a جديدة) أو نريد تغليف جميع العناصر a p بالوسم div ، يمكنك أن تنظر إلى الشكل التالي لتتضح الصورة :



تؤمن لنا مكتبة **jQuery** هذه الإمكانية عبر مجموعة من الدوال فالدّالّة **wrap** تقوم بتغليف كل عنصر من مجموعة العناصر التي تطبّق عليها بالغلاف الممرّر لها كوسيط **parameter** و لهذه الدالة الشكل التالي :

```
$('selector').wrap('Wrapper');
```

حيث تقوم بتغليف كل من العناصر المحددة بالمحدد **selector** بالغلاف **Wrapper** و لو أردنا تحويل الشكل التوضيحي السابق إلى شيفرة **jQuery** لكتبنا مثلاً :

```
$('img').wrap('<a href="#"></a>');
```

و بالفعل فإن الدالة السابقة تقوم بتغليف كل عنصر صورة بالمستند بعنصر الرابط الممرر لها بمعنى أن يصبح هناك غلاف لكل عنصر من المجموعة المحددة و لو أردنا تغليف جميع العناصر المحددة بغلاف واحد فقط لأتى دور الدالة **wrapAll** التي تلبي رغبتنا تماماً و لها الشكل :

```
$('selector').wrapAll('Wrapper');
```

و المثال التالي يوضّح عمل الدالّة الأولى **Wrap** :

```
<html>
  <head>
    <script src="jquery.js" type="text/javascript">
    </script>
    <script type="text/javascript">
      $(document).ready(function() {
        $('img[src]').wrap('<a href="#"></a>');
```

```
      });
    </script>
  </head>
  <body>
    <img src="noPic.gif" alt="no Pic!" />
  </body>
</html>
```

إذ أنَّه عند استعراض هذه الصفحة سيتم تغليف الصُّورة بالرّابط و هذا تحقيق فعلي للشكل التوضيحي الذي عُرِضَ في بداية الفقرة .

في حالات أخرى قد نحتاج إلى تغليف محتوى مجموعة من العناصر بدلاً من تغليفها نفسها و هذا ما تقدِّمه الدالة `wrapInner` التي تستخدم بالشكل :

```
$('selector').wrapInner('Wrapper');
```

<u>ملحوظة</u> : يمكن الاستفادة من دوال التغليف لنسخ العناصر من مكان إلى آخر ضمن المستند و ذلك عن طريق جعل الغلاف الجديد عنصراً من العناصر الموجودة أصلاً في المستند .

دوال حذف عناصر المستند :

توفر مكتبة **jQuery** دوالاً أخرى خاصة بحذف عناصر المستند فالدالة **remove** التي تستعمل بالشكل التالي :

```
$('selector').remove();
```

تقوم بحذف جميع العناصر التي يعيدها المحدد **selector** ، أما في حال الرغبة بحذف المحتوى الداخلي **inner content** الخاص بهذه العناصر فقط و الإبقاء عليها فيمكن استعمال الدالة **empty** التي لها الشكل :

```
$('selector').empty();
```

<u>ملحوظة :</u> هناك طريقة أخرى لنسخ عناصر المستند تؤمّنها الدّالّة **clone** التي تُستَعمل غالباً مع دوال **before** و **after** و **append** بصيغة تشبه المثال التالي الذي يقوم بنسخ عنصر **div** من المستند و يضيفها إلى نهاية **div** آخر

```
$('#sourceDiv').clone().appendTo('#targetDiv');
```

دوال التعامل مع عناصر النماذج FORM ELEMENTS :

تعتبر النماذج **Forms** من أهم العناصر في صفحات الـ **Web** لما لها من قيمة في تفاعل المستخدم مع تطبيق الـ **Web** الخاص بك و لكنَّ هذه الأهمية تأتي على حساب صعوبة في التعامل معها برمجياً من طرف العميل و يتجلّى ذلك بوضوح من خلال عدة سيناريوهات و لعلَّ هذا ما دفع الهيئة العالمية **W3C** لدعوة مبرمجي **java script** إلى الحذر أثناء التعامل معها و لهذا و ذاك تؤمن مكتبة **jQuery** مجموعة خاصة من الدَّوال للتعامل مع عناصر

النماذج و تفادي أغلب المشاكل الشائعة بأسهل الطرق و هنا نستهلُّ حديثنا عن هذه الدَّوال بالحديث عن الدّالّة val التي تستخدم لقراءة القيم من عناصر النماذج و لها الشكل التالي :

```
$('selector').val();
```

تعيد هذه الدالة قيمةً وحيدةً في حال كان المحدد selector يعيد عنصراً وحيداً هي قيمة ذلك العنصر ، أمّا في حال كون المُحدِّد selector يعيد أكثر من عنصر فإن الدالة val تعيد قيمة أوَّل عنصر من هذه العناصر .

ملحوظة : **يقصد بقيمة العنصر هنا القيمة التي تعرّفها الواصفة value لذلك العنصر و من الجدير بالذكر هنا أن الدّالّة val في حالة استدعائها على عنصر تحديد يسمح باختيار أكثر من خيار في وقت واحد Multi-select element فإنّ الدالّة val تعيد مصفوفةً تمثل الخيارات المنتقاة من قبل المستخدم .**

و كمثال على استعمالها فإن كتابة سطر كالتالي :

```
$('#firstName').val();
```

يعيد قيمة الواصفة value للعنصر ذو المعرف firstName .

على الرغم من أن الدالّة val تحلُّ الكثير من مشاكل التعامل مع العناصر إلا أنه يجب تذكّر نقطتين هامتين عند التعامل معها : النقطة الأولى أنه إن كان أول عنصر من المجموعة التي يعيدها المحدد selector ليس عنصر نموذج فإن خطأً برمجياً سيظهر في مستند الـ Web ، النقطة الثانية هي أن الدالة val تعيد قيمة الواصفة value لعناصر صناديق الاختيار check box و radio button بغض النظر عن كون هذه العناصر في حالة

selected أو لا و لكنَّ حلاً لهذه المشكلة يتوفر بسهولة بعد مراجعة جدول المحدِّدات و كتابة شيء شبيه بما يلي :

```
$('[name=radioGroup]:selected').val();
```

للدّالّة val استخدام آخر غير قراءة القيم من عناصر النماذج و هو كما تتوقع إسناد القيم لها و للقيام بهذه العملية نستعمل الشكل :

```
$('selector').val(value);
```

كما تستخدم الدالة val للقيام بتغيير حالة عناصر radio buttons و check boxes و عناصر <select> إلى حالة التحديد selected و ذلك عبر الصيغة :

```
$('selector').val(values);
```

حيث تقوم الدالة بالبحث ضمن العناصر التي يعيدها المحدد selector عن العناصر التي تحمل إحدى القيم المذكورة في مصفوفة القيم values الممررة للدالة و تجعل حالتها selected ، فالجملة التالية مثلاً :

```
$('input').val(['abokamal','jQuery','Mukhtar']);
```

تجعل الحالة selected حالةً لجميع عناصر النماذج التي تحمل واصفة value الخاصّة بها إحدى القيم abokamal أو jQuery أو Mukhtar و المثال التالي يوضِّح هذا الاستخدام:

```
<html>
```

```
  <head>
    <script src="jquery.js" type="text/javascript">
    </script>
    <script type="text/javascript">
      $(document).ready(function() {
        $('*').val(['abokamal', 'jQuery']);
      });
    </script>
  </head>
  <body>
    <select id="Select1">
      <option value="X">Abokamal</option>
      <option value="jQuery">jQuery</option>
      <option>Mukhtar</option>
    </select>
  </body>
</html>
```

ملحوظة : قد يكون هناك بعض السيناريوهات التي لا تلبي فيها الدالة val جميع متطلباتك و حينها يمكنك تجربة استعمال الإضافة plugin ذات الاسم Form Plugin و التي سنمرُّ عليها في الدقائق الخاصة بالإضافات .

من الدقيقة 57 إلى الدقيقة 75 :

الأحداث

Events

الأحداث Events :

قبل أن نناقش استثمار jQuery لتسهيل التعامل معها علينا أن نعرف شيئاً بسيطاً عنها ، ما هي الأحداث ؟

حسناً .. تمتلك كلمة حدث Event المعنى نفسه في مختلف تقنيات البرمجة سواءً أكنا نبرمج تطبيقاً لسطح المكتب أو تطبيقاً للـ Web أو تطبيقاً للهواتف الذكية أو لغيرها و كالمعنى اللغوي لاسمه يقصد بالحدث حدوث أمر ما أو تغيير معيّن يشعر به التطبيق و يستجيب له أحياناً -و هذا هو الغالب- أو يتجاهله في أحيان أخرى و هناك عدّة أنواع من الأحداث منها ما يسببه نظام التشغيل مثل بدء تشغيل التطبيق أو إنهاء تشغيله أو مقاطعته بشكل معين و منها ما يسببه المستخدم و هذا ما يهمنا صراحةً و من الأحداث التي يسببها المستخدم حدث النقر على زر معين في التطبيق Click أو حدث النقر المزدوج Double click أو حدث مرور الفأرة فوق منطقة ما من التطبيق Mouse move ... إلخ.

التطبيقات بطبيعتها لا تستجيب للأحداث و لكنها تشعر بحدوثها بمعنى أن تطبيقك يعرف تماماً أنّك قمت بالنقر على الزر الفلاني لحظة النقر كما يعرف تماماً أنك قمت بالكتابة في مربع النص لحظة الكتابة و لكنه بشكل افتراضي يتجاهل هذه الأحداث و لا يستجيب لها إلا إن أجبره المبرمج على ذلك بشكل صريح عن طريق ما يعرف بالتعامل مع الأحداث أو Event handling ، و لأن هذا الكتاب يسعى لتبسيط الأمور و يدّعي ذلك فيكفي أن أقول على سبيل التبسيط أنه للتعامل مع حدث معين -و الكلام هنا عام و ليس في مجال الـ Web فحسب- يكفي أن تكتب دالةً function تقوم بعملية معينة ثم تقوم بربط هذه الدالة بالحدث و عندها تصبح استجابة تطبيقك عندما يتم تفجير الحدث (عندما يقع الحدث) تنفيذ الدالة المرتبطة به.

في مجال الـ **Web** المشكلة العظمى التي كانت تواجه المبرمجين و مطوري مواقع الانترنت هي مشكلة اختلاف المتصفحات **Browsers** حيث أن كل متصفح يمتلك طريقته الخاصة في التعامل مع تطبيقات الـ **Web** و عند الحديث عن الأحداث فالأمر يطول و يطول ليستطيع المطور منا كتابة شيفرة **Java Script** تعمل على جميع المتصفحات ، مع **jQuery** نلقي بكل هذه المشاكل خلف ظهورنا و نستعمل دوال المكتبة لكتابة شيفرات عصرية تعمل لدى الجميع بضمان كبرى شركات البرمجيات في العالم.

في المعايير الموحدة لـ **HTML** و **XHTML** هناك مجموعة من الأحداث التي لها استجابات افتراضية لا تتطلب منا كتابة أي شيفرة فمثلاً عند تفجير حدث النقر المفرد **Click** على رابط **Link** فإن المتصفح ينقلنا إلى الصفحة الجديدة دون أي تدخل منا لأن هذه الاستجابة موجودة ضمنياً و هذا هو الحال مع بعض الأحداث الأخرى و بالطبع يمكن تجاوز هذه الاستجابات الافتراضية عند الحاجة ، أمَّا بالنسبة للطريقة التقليديّة للتعامل مع الأحداث فإنَّها تقتضي أن نستعين بواصفات خاصة توضع مع الوسوم لربط حدث معيّن بدالة معينة ، أغلب أسماء هذه الواصفات تبدأ بحرف الجر **on** فمثلاً من أجل حدث النقر **Click** هناك واصفة اسمها **onclick** و من أجل حدث مرور الفأرة **mouse Move** هناك واصفة اسمها **onmousemove** و من أجل حدث الانتقال إلى عنصر **focus** هناك واصفة اسمها **onfocus** و هكذا .. و لعلَّ المثال التالي يوضح الصّورة علماً أنه مكتوب بطريقة تقليدية دون الاعتماد على المكتبة الرائعة **jQuery** :

```
<html>
  <head>
    <script type="text/javascript">
      function myClickHandler() {
        alert("تم النقر على الزر") ;
```

```
      };

      function myMoveHandler() {
         alert("تم مرور مؤشّر الفأرة فوق الصورة") ;
      };
    </script>
  </head>
<body>
<input id="Button1" type="button" value="انقرني"
onclick="myClickHandler()" />
<img src="myPic.gif" onmousemove="myMoveHandler()" />
  </body>
</html>
```

الصفحة البسيطة تحتوي زراً و صورةً و لو انتبهت لواصفات الزر لرأيت الواصفة **onclick** و لو دققت في قيمتها قليلاً لوجدتها اسماً لدالة من دوال **java script** الموجودة في المستند ألا وهي الدالة **myClickHandler** ، كما أنك لو دققت في واصفات الصورة لرأيت الواصفة **onmousemove** التي تحتوي على اسم لدالة أخرى من دوال المستند و بنظرة بسيطة نعرف أن الدالة الأولى ستُستدعى عند النّقر على الزّر و الثانية ستُستدعى عند مرور مؤشر الفأرة فوق الصورة و لكَ أن تجرب .

حسناً .. المثال السابق يعتبر أبسط أمثلة التعامل مع الأحداث على الإطلاق و يمكن اتباع طريقة المثال في التعامل مع الأحداث و استخدام أوامر **jQuery** ضمن جسم الدالة التي تُستدعى عند تفجير الحدث و لكن الطريقة الأفضل هي استخدام نموذج أحداث **jQuery**.

تؤمن مكتبة **jQuery** طريقةً سهلةً للتعامل مع الأحداث عبر مجموعة من الدوال منها الاحترافية التي سأناقشها بعد المجموعة الأخرى البسيطة التي تقابل كل دالة منها واصفةً من الواصفات الخاصة بالأحداث لكن بعد الاستغناء عن حرف الجر **on** في الاسم ، فمثلاً الدالة البسيطة التي تعين الدالة التي ستستجيب للحدث **click** اسمها **click** و ليس **onclick** و أختها التي تعين الدالة التي ستستجيب لحدث مرور الفأرة اسمها **mousemove** و ليس **onmousemove** و كمثال على استعمالها نستطيع إعادة كتابة الصفحة السابقة بالشكل :

```
<html>
  <head>
    <script type="text/javascript"  src="jquery.js">
    </script>
    <script type="text/javascript">
      $(document).ready(function(){
          $('#Button1').click(function(){
            alert("تم النقر على الزر");
          });
          $('img').mousemove(function(){
            alert("تم مرور المؤشر فوق الصورة");
```

```
            });
        });
  </script>
  </head>
  <body>
    <input id="Button1" type="button" value="انقرني" />
    <img src="myPic.gif" />
  </body>
</html>
```

عمل الصفحة أعلاه مطابق تماماً لعمل الصفحة القديمة لكن باستخدام jQuery و من أوّل الفوائد الظاهرة هنا دون تفكير هي سهولة الكتابة مقارنةً بالطريقة القديمة إضافة إلى زيادة تنظيم و ترتيب المستند و لو انتبهت إلى الوسوم الآن لرأيت أنّه لم يعد هناك واصفات خاصة بالأحداث و بهذا تكون قد نجحت في فصل التصميم عن الشيفرة البرمجية الخاصة بالملف منطقياً كما يمكن وضع الشيفرة البرمجية كاملةً في ملف منفصل و من ثم تضمينه ضمن الصفحة كما يفعل المبرمجون الجادّون و بهذا يكون هنالك فصل فيزيائي حتى !

و بما أن شكل استدعاء دوال الأحداث البسيطة في jQuery نفسه لجميع هذه الدّوال فلا داعيَ لذكر كل واحدة منها بالتفصيل و نكتفي بذكر أسماء هذه الدوال و شرحها في الجداول التالية :

دوال الأحداث الخاصة بالنافذة (المستند ككل) :

اسم دالة الحدث	لحظة تفجير الحدث
load	عند تحميل المستند
unload	عند إغلاق المستند

دوال الأحداث الخاصة بعناصر النماذج :

اسم دالة الحدث	لحظة تفجير الحدث
blur	عندما خسارة العنصر للتركيز (lose focus) هو معاكس تماماً للحدث focus
change	عند حدوث تغيير على العنصر
submit	عند إرسال القيم إلى الخادم
focus	عند انتقال التركيز إلى العنصر (مثلاً في حالة الانتقال بين مجموعة من حقول الإدخال فإن الحدث focus يقع على حقل الإدخال التالي بمجرد الانتقال إليه عن طريق مفتاح الجدولة Tab مثلاً)
select	عند اختيار بند من محتويات العنصر (نرى هذا الحدث في مثل حالة العنصر (<select>

scroll	عند تحريك أشرطة التمرير (نرى هذا الحدث في مثل حالة العنصر **textarea**)

دوال الأحداث الخاصة بلوحة المفاتيح :

اسم دالة الحدث	لحظة تفجير الحدث
keydown	عند الضغط على مفتاح من لوحة مفاتيح
keypress	بعد الضغط على مفتاح من لوحة المفاتيح و تحريره
keyup	عند تحرير مفتاح من لوحة المفاتيح

دوال الأحداث الخاصة بأزرار الفأرة :

اسم دالة الحدث	لحظة تفجير الحدث
click	عند النقر المفرد
dblclick	عند النقر المزدوج
mousedown	عند الضغط على زر الفأرة
mousemove	عند مرور مؤشر الفأرة
mouseout	عند خروج مؤشر الفأرة (يقصد بدخول مؤشر الفأرة أول مرور له فوق العنصر و يقصد بالخروج لحظة ابتعاده عنه)

mouseover	**عند دخول مؤشر الفأرة**
mouseup	**عند تحرير زر الفأرة (انتهاء الضغط)**

دوال الأحداث الأخرى :

اسم دالة الحدث	**لحظة تفجير الحدث**
error	**عند حدوث خطأ**
resize	**عند تغيير الحجم**

تذكّر : جميع الدّوال المذكورة في الجداول أعلاه لها الشكل العام التالي

```
$('selector').event(function);
```

حيث تقوم بتعيين الدالة function استجابةً للحدث event لجميع العناصر التي يعيدها المحدد selector .

و الآن و بعد أن رأينا سهولة دوال الأحداث البسيطة ما رأيك في أن نرى مرونة الدوال الأخرى الخاصة بالأحداث ؟

حسناً .. تقوم الدالة bind بتحديد دالة معينة كاستجابة لحدث محدد و لها الشكل العام التالي :

```
$('selector').bind('event' , function);
```

حيث أن الوسيط **event** هو اسم الحدث و عادةً ما يكون واحداً من الأحداث ذاتها المذكورة في الجداول السابقة ، أما الوسيط **function** فهو اسم الدالة التي ستنفذ عند تفجير الحدث و إن تعليمتي **jQuery** التاليتين تقومان بنفس العمل تماماً :

```
$('img').click(myFunction);

$('img').bind('click' , myFunction);
```

و من الجدير بالذكر أن عمل **bind** أكثر من مرة على نفس العنصر يؤدي إلى تنفيذ جميع الدوال التي تربطها بالعنصر عند تفجير الحدث و هذا ممكن أيضاً باستخدام الدوال البسيطة لذا فإن التعليمة التالية :

```
$('img').bind('click',F1).bind('click',F2).bind('click',F
3);
```

تقوم باستدعاء الدوال **F1** و **F2** و **F3** بالترتيب عند تفجير حدث النقر الخاص بعناصر الصور في المستند ، و إن التعليمة التالية تقوم بالمثل أيضاً :

```
$('img').click(F1).click(F2).click(F3);
```

و على النقيض تماماً من عمل الدالة **bind** تقوم الدالة **unbind** بإلغاء ربط أحد الدوال بحدث معين و لها نفس الشكل الخاص بـ**bind** تماماً لذا فإن التعليمتين التاليتين :

```
$('img').bind('click',Func1).bind('click',Func2)
.bind('click',Func3);

$('img').unbind('click',Func2);
```

ستقومان باستدعاء الدالتين **Func3** و **Func1** عند تفجير حدث النّقر الخاص بعناصر الصور في المستند دون تطبيق استدعاء الدالة **Func2** لأن الدالة **unbind** قامت بإلغاء ربطها بالحدث .

كما توفر مكتبة **jQuery** دالة خاصة مشابهة للدالة **bind** اسمها الدالة **one** الفرق الوحيد بينها و بين الدالة **bind** أن الأخيرة تستدعي مجموعة من الدوال عند تفجير حدث واحد فقط بينما تستطيع الدالة **one** أن تستدعي مجموعة من الدوال عند تفجير مجموعة من الأحداث و لذا فإن التعليمة التالية :

```
$('img').one("click , mousemove , dblclick", '
myFunction');
```

تقوم باستدعاء الدالة **myFunction** عند تفجير أيٍّ من الأحداث **click** أو **mousemove** أو **dblclick** !

و الآن بعد أن فهمنا نموذج الأحداث الخاص بالمكتبة **jQuery** يجب أن نذكر أن جميع دوال الأحداث يمكن أن تزوّد بوسيط اختياري يُنشَأ تلقائياً بواسطة المكتبة ، يحمل هذا الوسيط معلوماتٍ هامّةً تخصُّ الحدث و يسمّى "الكائن **event**" أو "الكائن **e**" اختصاراً ، و لإنشائه نستعمل نفس الصيغة الخاصّة بدوال الأحداث لكن مع تمرير وسيط إضافي يمثل الوسيط **e** و كمثال على ذلك يمكن أن نكتب :

```
$('a').click(function(e){

     Alert("مرحباً");

});
```

لا يختلف عمل الأسطر أعلاه عن عملها بدون تمرير الوسيط **e** أبداً فهي تقوم بعرض الرسالة "مرحباً" عند النقر على أي رابط في الصفحة و لكن ما يميزها أن وسيطها **e** يحمل قيماً خاصة

تمثل معلومات عن الحدث **click** يمكن الاستفادة منها في جسم الدالة فمثلاً يمكن أن نكتب :

```
$('a').click(function (e){
     alert(e.type);
});
```

و سيكون نتيجة النقر على الروابط هذه المرة هو عرض رسالة تحتوي على الكلمة **click** و هي هنا قيمة التعبير **e.type** (في الحقيقة قيمة الخاصية **type** الخاصة بالكائن **e**).

<u>ملحوظة :</u> قد لا يبدو الأمر ذا أهمية في البداية لكن لو درست التعقيدات الخاصة بهذا الكائن في **java script** و اختلاف طرق التعامل معه باختلاف المتصفح لعرفت العمل الكبير الذي قام به فريق تطوير المكتبة **jQuery** .

لا يفوتنا أن نذكر هنا أنه على الرغم من أن الصيغة ذاتها تستخدم لتعريف الكائن **e** في جميع دوال الأحداث إلا أن خاصيات الكائن **e** تختلف من حدث إلى آخر و هذا طبيعي إذ أنَّ المعلومات المتعلقة بحدث معيّن قد لا تعني شيئاً بالنسبة لحدث آخر و كمرجع سريع نلخّص أهم خاصيّات الكائن **e** في هذا الجدول :

الخاصية	المعلومة التي تحتويها
altKey	تحتوي القيمة **true** في حال كون المفتاح **alt** مضغوطاً لحظة تفجير الحدث و تحتوي **false** في الحالة المعاكسة
ctrlKey	تحتوي القيمة **true** في حال كون المفتاح **ctrl** مضغوطاً لحظة تفجير الحدث و تحتوي **false** في الحالة المعاكسة

keyCode	تتعلق بالحدثين keyup و keydown و تعيد هذه الخاصية قيمة الـ ASCII الخاص بالمفتاح المضغوط (و في حال الضغط على حرف ما فإنها تعيد قيمة الحرف الكبير-كابيتال لتر- أو upper case فمثلاً هذه الخاصية ستحتوي القيمة 65 في حالة ضغط المفتاح a أو ضغط المفتاح A)
pageX	تتعلق بأحداث الفأرة و تعيد الإحداثيات الأفقية X الخاصة بمؤشر الفأرة بالنسبة للصفحة (بعد مؤشر الفأرة عن حافة الصفحة اليسرى عادةً)
pageY	تتعلق بأحداث الفأرة و تعيد الإحداثيات العمودية Y الخاصة بمؤشر الفأرة بالنسبة للصفحة (بعد مؤشر الفأرة عن حافة الصفحة العلوية عادةً)
screenX	تتعلق بأحداث الفأرة و تعيد الإحداثيات الأفقية X الخاصة بمؤشر الفأرة بالنسبة للشاشة (بعد مؤشر الفأرة عن حافة الشاشة اليسرى عادةً)
screenY	تتعلق بأحداث الفأرة و تعيد الإحداثيات العمودية Y الخاصة بمؤشر الفأرة بالنسبة للشاشة (بعد مؤشر الفأرة عن حافة الشاشة العلوية عادةً)
relatedTarget	تتعلق ببعض أحداث الفأرة و تعيد معرف العنصر الذي دخل إليه / خرج عنه مؤشر الفأرة لحظة تفجير الحدث
shiftKey	تحتوي القيمة true في حال كون المفتاح shift مضغوطاً

	لحظة تفجير الحدث و تحتوي **false** في الحالة المعاكسة
type	تستعمل مع جميع الأحداث و تعيد اسم الحدث
which	بالنسبة لأحداث لوحة المفاتيح تعيد هذه الخاصية قيمة ال **ASCII** الخاصة بالمفتاح المضغوط ، أما بالنسبة لأحداث الفأرة تعيد رقماً يمثل زر الفأرة المضغوط حيث أن الرقم 1 يمثل زر الفأرة الأيسر و الرقم 2 يمثل زر الفأرة الأوسط و الرقم 3 يمثل الزر الأيمن

آخر ما يجب ذكره في هذه الدقائق هو أن المكتبة jQuery تعطينا إمكانية إلغاء الاستجابة الافتراضية للأحداث التي تملك استجابة افتراضية مثل حدث النقر على رابط معين إذ أنه يملك استجابة افتراضية تنقلنا إلى العنوان الذي يشير إليه ، و تسمح المكتبة بإلغاء الاستجابة الافتراضية عن طريق الدالة preventDefault الخاصة بالكائن e و التي تستخدم كمايلي :

```
$('a').click(function(e){

     e.preventDefault();

});
```

و الآن نحن متأكدون أن الاستجابة الافتراضية لحدث النقر على الروابط في الصفحة لن تحدث .

من الدقيقة 75 إلى الدقيقة 87 :

الحَركات

Animations

الحَركات Animations :

أما زلتَ تذكر ما قلتُ لك عند عرضنا لموقع **Microsoft** كمثال في مقدِّمة هذا الكتاب ؟

حسناً .. سأذكّركَ أنّني قلتُ لكَ : "لا تدع الجمال يخدعك" و قلت : "لا تعتقد أن ما تراه من عمل فلاش **Flash**" و قلت في النهاية : "أغلب ما تراه من عمل المكتبة **jQuery**" قلت كل ما سبق هكذا دون أي برهان و الآن حان الوقت لنبرهن ذلك سوياً .

المؤثرات البسيطة :

تؤمّن المكتبة **jQuery** مجموعة من الدّوال الخاصّة بالحركات **Animations** و لعلّ أبسط أنواع الحركات تكمن في إظهار و إخفاء العناصر إذ تؤمّن المكتبة هاتين الوظيفتين البسيطتين عن طريق الدّالّة **show** التي تُستعمل للإظهار و الدّالّة **hide** التي تُستعمل للإخفاء و تستعمل الدالتان على الشكل :

```
$('selector').show();
```

حيث يقوم هذا السطر بإظهار العناصر المخفية التي يعيدها المحدد **selector** في حين أن التعليمة :

```
$('selector').hide();
```

تقوم بإخفاء العناصر التي يعيدها المحدد **selector** و كثيراً ما يستعمل المبرمجون هاتين الدالتين في إنشاء القوائم الشجرية **Tree Lists** في شيء شبيه بالمثال التالي:

```
<html>
<head>
  <script src="jquery.js" type="text/javascript">
```

```
  </script>

  <script type="text/javascript">

    $(document).ready(function(){

      $('li:has(ol)').css('curosr','pointer');

      $('li:has(ol)').click(function (){

        If($(this).children().is(':hidden'))

          $(this).children().show();

        else

          $(this).children().hide();

      });

    });

  </script>

</head>

<body dir="rtl">

<fieldset>

<legend>مثال القائمة الشجرية</legend>

<ul>

 <li>العنصر الأول</li>

     <li>العنصر الثاني

       <ol>
```

```
    <li>عنصر فرعي</li>
    <li>عنصر فرعي</li>
    <li>عنصر فرعي</li>
    <li>عنصر فرعي</li>
    <li>عنصر فرعي</li>
  </ol>
</li>
<li>العنصر الثالث</li>
<li>العنصر الرابع
  <ol>
    <li>عنصر فرعي</li>
    <li>عنصر فرعي</li>
    <li>عنصر فرعي</li>
  </ol>
</li>
<li>العنصر الخامس</li>
<li>العنصر السادس</li>
</ul>
```

```
 </fieldset>

</body>

</html>
```

تحتوي الصفحة أعلاه قائمة غير مرتبة `Unordered list` من العناصر ممثلة بالوسم `ul` ، يحتوي بعض عناصر هذه القائمة على قوائم مرتبة فرعيَّة `orederd list` من العناصر ممثلة بالوسوم `ol` .

يقوم السطر :

```
$('li:has(ol)').css('curosr','pointer');
```

بتغيير شكل مؤشر الفأرة إلى الشكل المعبّر عن إمكانية النقر و ذلك للعناصر التي يعيدها المحدد (`li:has(ol)` و الذي يعني عناصر الوسوم `li` التي تحتوي على وسوم `ol` فرعية (راجع جدول المحدِّدات) .

بينما تقوم الأسطر التالية بإضافة الوظيفة الخاصة بالقائمة الشجرية إلى تلك العناصر :

```
$('li:has(ol)').click(function(){

  if($(this).children().is(':hidden'))

    $(this).children().show();

  else

    $(this).children().hide();

});
```

إذ تقوم ببناء روتين (دالة) خاص بحدث النقر على تلك العناصر يقوم بإخفائها إنْ كانت ظاهرة و إظهارها إنْ كانت مخفية و هنا يجدر التنويه أن الدالة **is** إحدى دوال المكتبة **jQuery** تقوم بإعادة القيمة **true** إن كانت العناصر المطبقة عليها توافق المحدّد الممرّر لها و تعيد **false** في الحالة المعاكسة ، بينما تقوم الدالة **children** بإعادة العناصر الأبناء للعنصر المطبقة عليه و بناءً على ما سبق يصبح معنى السطر التالي :

```
if($(this).children().is(':hidden'))
```

"إذا كان أبناء العنصر **this** غير ظاهرين" ، و لا يخفى عليك إن كنت قد قرأت الفصول السابقة أن العنصر **this** هنا يعني العنصر الذي وقع عليه حدث النقر.

تؤمّن المكتبة **jQuery** دالةً للقلب في مثل هذه الحالة تشبه دالة القلب في حالة أوراق الأنماط اسمها **toggle** و تقوم هذه الدالة بإخفاء العنصر إن كان ظاهراً أو إظهاره إن كان مخفياً و لها الشكل التالي :

```
$('selector').toggle();
```

حيث تقوم الدالة بقلب حالة ظهور العناصر التي يعيدها المحدد **selector** و يمكننا تحسين مثال القائمة الشجرية بالاعتماد على هذه الدالة و اختصار شيفرة **jQuery** الخاصة به لتصبح كما يلي :

```
$('li:has(ol)').click(function(){
    $(this).children().toggle();
});
```

و هنا يظهر بوضوح صدق كلام فريق تطوير المكتبة jQuery عندما طرح شعارها "اكتب أقل .. افعل أكثر" .

و قد يقول قائل بعد قراءة الأسطر أعلاه : "نستطيع عمل هذا ببساطة ! أين الحركة ؟" و هنا يجدر بنا أن نعترف أنَّ الدّوال show و hide و toggle عندما تُستدعى بدون أي وسيط فإنّها لا تعمل كحركاتٍ أبداً و إنّما تقوم بإخفاء و إظهار العناصر بشكلٍ جاف ، و يمكن تحسين هذا الشكل بتمرير وسيطٍ إلى كلٍّ من هذه الدوال يمثّل سرعة الحركة و عليه يصبح شكل الدوال كمايلي :

```
$('selector').show(speed);

$('selector').hide(speed);

$('selector').toggle(speed);
```

حيث أن الوسيط speed يمثل سرعة تنفيذ المؤثر (زمن تنفيذ المؤثر في الحقيقة) و يمكن التعبير عن الوسيط speed بصيغتين :

أ - صيغة نصّية : و هي قيمة من القيم 'slow' التي تعني تنفيذاً بطيئاً للمؤثر أو 'normal' و التي تعني تنفيذ المؤثر بالسّرعة الطبيعية أو 'fast' و التي تعني تنفيذ المؤثر بشكل سريع.

ب - صيغة رقمية : تمثل زمن تنفيذ المؤثر بالمللي ثانية (0.001 ثانية) و بالاستفادة من هذه المعلومة يمكن تحسين شيفرة jQuery في مثال القائمة الشجرية لتصبح كمايلي إذا عبرنا عن السرعة بالصيغة الرقمية :

```
$('li:has(ol)').click(function(){
```

```
    $(this).children().toggle(1000);
});
```

أو كمايلي إذا عبرنا عن السرعة بالصيغة النصية :

```
$('li:has(ol)').click(function(){
    $(this).children().toggle('slow');
});
```

و الآن أصبح كل من الإخفاء و الإظهار يستغرق زمناً قدره ثانية واحدة (1000 مللي ثانية) و في هذه الحالة يبدأ جمال حركات **jQuery** بالظهور .

آخر ما يجب الاعتراف به في سياق الحديث عن الدوال **show** و **hide** و **toggle** هو أن لها شكلاً آخراً هو الشكل :

```
$('selector').show(speed,F);
$('selector').hide(speed,F);
$('selector').toggle(speed,F);
```

حيث أن الوسيط **F** هو اسم دالة أخرى موجودة في الصفحة يتم تنفيذها عند اكتمال تنفيذ المؤثر .

مؤثرات التلاشي :

و الآن يمكننا الانتقال للحديث عن دوال أخرى من دوال حركات **jQuery** مثل دالة الظهور المتلاشي **Fade in** التي لها الشكل التالي :

```
$('selector').fadeIn(speed,F);
```

حيث أن الوسيط الاختياري **speed** يمثل سرعة تنفيذ المؤثر و ككل وسطاء السرعة في دوال الحركات الخاصة بمكتبة **jQuery** يأخذ هذا الوسيط قيمة من القيم النصية **slow** أو **normal** أو **fast** أو قيمةً رقمية تعبر عن زمن التنفيذ بالمللي ثانية ، في حين أن الوسيط الاختياري **F** يمثل اسماً لدالة موجودة في المستند يتم استدعاؤها بعد انتهاء تنفيذ المؤثر.

تذكّر: عندما نقول عن وسيط ما أنه وسيط اختياري **Optional parameter** فنحن نعني أننا نستطيع عدم تمريره عند استدعاء الدالة.

و على عكس دالة الظهور المتلاشي تقوم دالة الاختفاء المتلاشي **fade out** بإخفاء العنصر بشكلٍ متلاشٍ و لها الصيغة التالية :

```
$('selector').fadeout(speed,F);
```

حيث أن الوسيط **speed** يمثل سرعة التنفيذ (كالعادة) و الوسيط **F** يمثل اسماً لدالة موجودة في المستند تستدعى لحظة انتهاء تنفيذ المؤثر (كالعادة أيضاً).

توفر المكتبة **jQuery** دالةً أخرى خاصة بالتلاشي اسمها **fadeTo** و هذه الدالة تقوم بتحديد درجة الشفافية **opacity** الخاصة بالعناصر التي يعيدها المحدد و لها الشكل التالي :

```
$('selector').fadeTo(speed,opacity,F);
```

حيث أن الوسيط **speed** يمثل سرعة التنفيذ و الوسيط **F** يمثل اسماً لدالة تستدعى لحظة انتهاء تنفيذ المؤثر أما الوسيط **opacity** يمثل درجة شفافية العنصر التي سينتقل إليها أثناء تنفيذ المؤثر و يستقر عليها بعد انتهاء تنفيذ المؤثر و يمكن أن تكون درجة الشفافية بين 0

(شفاف لدرجة عدم الظهور) و 100 (ظهور بشكل طبيعي) ، و يجدر الإشارة هنا أن الدالة **fadeTo** لا تقوم بحذف العناصر عند انتهاء الحركة كما تفعل **fadeOut** مثلاً .

مؤثّرات الانزلاق :

يتوفر في مكتبة **jQuery** دوال خاصة بحركات أجمل مثل دوال حركات الانزلاق و منها حركة الانزلاق السفلي **slide down** التي توفرها الدالة **slideDown** ذات الشكل :

```
$('selector').slideDown(speed,F);
```

حيث أن الوسيط **speed** يمثل السرعة و الوسيط **F** يمثل اسماً لدالة تُستدعى عند انتهاء المؤثر ، كما توفّر المكتبة دالةً للانزلاق العلوي **slide up** اسمها **slideUp** و لها الشكل :

```
$('selector').slideUp(speed,F);
```

حيث أنّ الوسيط **speed** و الوسيط **F** يمثلان سرعة التنفيذ و اسم الدالة التي تُستدعى عند الانتهاء من الحركة.

و تقدم المكتبة دمجاً للدالتين السابقتين بدالة قلب خاصة بهما اسمها الدالة **slideToggle** تقوم بإظهار العناصر المخفية باستخدام مؤثر **slide Down** و إخفاء العناصر الظاهرة باستخدام مؤثر **slide Up** و لها الشكل التالي :

```
$('selector').slideToggle(speed,F);
```

و لا داعيَ لنتذكّر معاً أنّ الوسيط **speed** يمثل سرعة التنفيذ و الوسيط **F** يمثل اسم الدالة التي تُستدعى عند انتهاء الحركة ☺.

إيقاف الحركة :

آخر دالة من دوال مكتبة **jQuery** الخاصة بالحركة هي الدالة **stop** و التي تستخدم ببساطة لإيقاف تنفيذ أي حركة قيد التنفيذ و تستخدم بالشكل :

```
$('selector').stop();
```

مرجع سريع لحركات المكتبة :

يلخص الجدول التالي أهم دوال الحركات الخاصة بمكتبة **jQuery** :

اسم الدالة	عملها	المجموعة
show	إظهار العناصر المخفية	مؤثرات بسيطة
hide	إخفاء العناصر الظاهرة	مؤثرات بسيطة
toggle	قلب حالة ظهور العنصر	مؤثرات بسيطة
fadeIn	ظهور متلاش	مؤثرات التلاشي
fadeOut	اختفاء متلاش	مؤثرات التلاشي
fadeTo	تلاشي إلى شفافية محددة	مؤثرات التلاشي
slideDown	انزلاق للأسفل	مؤثرات الانزلاق
slideUp	انزلاق للأعلى	مؤثرات الانزلاق
slideToggle	انزلاق إلى الحالة المعاكسة	مؤثرات الانزلاق

إنشاء حركات خاصة :

تسمح لنا مكتبة **jQuery** بإنشاء حركات خاصة غير تلك الافتراضية التي توفرها عبر الدالة **animate** و التي تمتلك الشكل التالي :

```
$('selector').animate(properties,speed);
```

حيث أن الوسيط **properties** هو كائن JSON (كالذي عرضناه سابقاً في الدقائق الخاصة بالدوال) يمثل مجموعة من الخصائص التي سيستقر عليها العنصر بعد انتهاء الحركة إذ أنَّ فكرة الحركات الخاصة تكمن في الانتقال من الحالة الطبيعية للعنصر إلى الحالة الجديدة التي تفرضها الخصائص المزودة بواسطة الوسيط **properties** ، أما الوسيط **speed** فإنه لا يزال يمثل سرعة الحركة و لا تزال القيم التي يستطيع أخذها كما ذكرناها سابقاً ، من المهم أن نذكر هنا أن الخصائص التي يمكن للكائن **properties** أن يحتويها هي خصائص الأنماط الخاصة بالعنصر و التي تأخذ قيماً رقمية فقط في العادة مثل **height** و **opacity** و **top** و **size-font** .. إلخ أو أن تأخذ أسماء إحدى دوال المكتبة الخاصة بالحركة مثل **show** و **toggle** و **hide** ... إلخ .

حسناً .. كمثال أوّل على إنشاء حركة خاصة بنا سنقوم بإكمال نقص في حركات المكتبة يكمن في الحركة **fadeToggle** التي لم نرَ أنها موجودة بشكل افتراضي على عكس باقي مجموعات الحركات التي تمتلك كل منها دالة قلب و لبناء هذه الحركة الخاصة يمكن أن نكتب شيئاً كهذا :

```
$('selector').animate({opacity : 'toggle'}, 'slow');
```

كما تلاحظ هنا فإن الوسيط **speed** هنا يحمل القيمة **slow** في حين أن الوسيط **properties** هنا هو كائن **JSON** التالي :

```
{opacity :'toggle'}
```

و الذي يعني قلب قيمة الخاصية **opacity** الخاصة بالعنصر في كل مرة .

و كمثال آخر يمكن إنشاء حركة خاصة بنا تقوم بمضاعفة حجم العناصر ثلاثة أضعاف كما يلي :

```
$('selector').animate(
{ width : $(this).width() * 3 ,
  height: $(this).height() *  3 }
, 'slow' );
```

و الآن أعتقد أن فكرة إنشاء الحركات الخاصة أصبحت واضحةً إذ ينبغي فقط أن تزود الدالة بمجموعة من القيم النهائية لخاصيات العنصر عبر كائن **JSON** و يمكنك تحديد السرعة عبر الوسيط الثاني **speed** .. حظاً موفقاً .

من الدقيقة 87 إلى الدقيقة 110 :

التخاطب مع الخادم

عبر تقنيّة AJAX

التخاطب مع الخادم عبر تقنية AJAX :

أستطيع أن أتطاول على التقنيات الخاصّة ببرمجة الـ **Web** لأقول : إنّ أي تقنية منهنَّ لم تستطع أن تفتح آفاق تطوير تطبيقات الـ **Web** كما فعلت تقنية **AJAX**.

AJAX اختصار لـ **A**synchronous **J**ava Script **A**nd **X**ML و هي تقنية تسمح لتطبيق الـ **Web** بالتخاطب مع الـ **Server** عبر إرسال طلبات غير متزامنة له تجري دون الحاجة لإعادة تحميل الصفحة مما يقرّب المسافة بين تطبيقات الانترنت و تطبيقات سطح المكتب و بالطبع فإنّ اختلاف مستعرضات الـ **Web** يجعل الأمور أكثر تعقيداً كعادته و كعادتنا سنلقي هذه التعقيدات خلف ظهورنا و نستعمل مكتبنا **jQuery** لتتولى أمر التعقيدات و تقدم لنا دوالاً بديلةً بسيطة تلبّي حاجاتنا تماماً و تسمح لنا بالتركيز على هدف تطبيقنا فقط دون أن نشغل بالنا بأي شيء آخر .

<u>ملحوظة :</u> قد تكون هذه الدقائق هي الأصعب في الكتاب إذ أنَّنا نناقش فيها استثمار المكتبة لتبسيط الشيفرات التي يكتبها مبرمجو **AJAX** لكنَّ هذا لا يعني أننا نغطّي التقنية **AJAX** و لا أي من تقنيات البرمجة من طرف الخادم فإذا لم تكن لديك خبرة في البرمجة من طرف الخادم **Server** أو كنتَ ممن لا يعرفون **AJAX** فباستطاعتكَ أن تقرأ عنهما قبل قراءة هذه الدقائق فهما موضوعان خارج إطار هذا الكتاب و قد يحتاجان أكثر من كتاب لتغطيتهما ، و إن كان الموضوع لا يهمّك تستطيع الانتقال مباشرةً إلى دقائق الإضافات .

<u>تذكّر :</u> الأمثلة في هذا الفصل تتطلَّب وجود تطبيق من طرف خادم بعيد أو محلّي .

جلب المحتوى من الخادم :

أوّل الدوال التي توفرها مكتبة jQuery لتأمين إرسال الطلبات غير المتزامنة للخادم Server هي الدالة load التي لها الشكل التالي :

```
$('selector').load(url,JSON,F);
```

حيث أن الوسيط url يمثل اسم الصفحة التي سيرسل إليها الطلب في الخادم ، في حين أنَّ الوسيط الاختياري JSON هو كائن يمثل مجموعة من الوسطاء التي سترسل إلى الصفحة ذاتها في الخادم أمّا الوسيط الاختياري F فهو يمثل اسماً لدالة موجودة في الصفحة تُستدعى لحظة انتهاء تنفيذ الطلب و بعد قدوم الاستجابة من الخادم و يمكن تمرير استجابة الخادم كوسيط لهذه الدالة ، و طبعاً أهم ما يجب ذكره أن الدالة load تقوم باستبدال محتوى الكائنات التي يعيدها المحدد Selector بالمحتوى الذي سيعود من الخادم Server نتيجة تنفيذ الطلب على شكل HTML (في الحقيقة ستعود الصفحة المطلوبة كاملةً) ، و كمثال على هذه الدالة قد نكتب شيئاً كمايلي :

```
$('#divA').load('myPage.aspx');
```

حيث ستقوم هذه الدالة باستبدال محتوى الوسم ذو المعرف divA بمحتوى الصفحة myPage.aspx الموجودة في نفس مسار صفحتنا الحالية.

و يمكننا التحكم بالعملية أكثر عن طريق تزويد الدالة بمحدد مع اسم الصفحة مما يسمح باستبدال المحتوى بالمحتوى المعاد من الخادم على شكل HTML و الذي يطابق المحدد فقط و عندها يمكننا كتابة شيء كمايلي :

```
$('#divA').load('myPage.aspx a[href=www.aw.com]');
```

حيث تقوم الشيفرة السابقة باستبدال محتوى الوسم ذو المعرف **divA** بالمحتوى الذي يطابق المحدد **a[href=www.aw.com]** و الذي ستعيده الدالّة **load** من الصفحة **myPage.aspx**.

و كمثال على تمرير وسطاء للصفحة الهدف في الخادم لنفرض أن الصفحة التي نريد استعادة قسم من محتواها تملك العنوان التالي :

/myPage.aspx?p_1=v_1&p_2=v_2

عندئذ يمكننا أن نكتب الشيفرة التالية :

```
$('#divA').load('/myPage.aspx', {p1:'v1', p2:'v2'});
```

حيث أن الوسيط الثاني هنا هو عبارة عن كائن **JSON** يمثل وسطاء الصفحة الهدف .

<u>ملحوظة :</u> من المهم أن نذكُر هنا أن المستعرض **Internet Explorer** يقوم بتخزين نسخة مؤقتة من الصفحات المطلوبة **Cache** لذا إنْ كانت الصفحة التي تنوي جلب معلوماتٍ منها متجددةً بشكل سريع مثل صفحات أسعار العملات مثلاً ، عندها احرص على تزويد وسيط ذا قيمة تتغير عشوائياً للدالة لأن هذا يجعل المستعرض سالف الذكر يعتبر كل طلب منها صفحةً مستقلة عن الأخرى و بالتالي لن يوثر عمل **Cache** لإحدى الصفحات على عمل المتبقي منهنَّ .

تمرير وسطاء للخادم من حقول النماذج :

غالباً ما نمرر إلى الدالة **load** قيماً من أحد نماذج الصفحة **Forms** كوسطاء للصفحة الهدف عبر استعمال دالة أخرى من دوال المكتبة هي الدالة **serialize** فمثلاً لاستدعاء الصفحة **myPage.aspx** مع وسطاء قيمهم قادمة من النموذج ذو المعرف **myForm** يمكن أن نكتب مايلي :

```
$('#divA').load('/myPage.aspx', $('#myForm').serialize());
```

و ستتولى الدالة **serialize** في هذا المثال توليد كائن **JSON** المناسب للدالة **load** و الذي سيمثل وسطاء الصفحة الهدف **myPage.aspx** .

ملحوظة : في حال تزويد قيمة للوسيط الاختياري **JSON** ستقوم الدالة بإرسال الطلب بالطريقة **POST** و في حال عدم تزويد قيمة لذات الوسيط ستقوم الدالة بإرسال الطلب بالطريقة **GET** .

إرسال طلبات من النوع GET :

تسمح لنا المكتبة **jQuery** بإرسال طلبات غير متزامنة من نوع محدد من النوعين **GET** و **POST** عبر مجموعة من الدوال منها الدالة **$.get** التي تقوم بإرسال طلب غير متزامن من النوع **GET** إلى الخادم و تعيد كائناً يمثل الاستجابة التي أعادها الخادم و لهذه الدالة الشكل التالي :

```
$.get(url,JSON,F);
```

حيث أن الوسيط **url** يمثل اسم الصفحة التي سيرسل إليها الطلب في الخادم و يمثل الوسيط الاختياري **JSON** مجموعة من الوسطاء التي سترسل إلى الصفحة ذاتها في الخادم أما الوسيط الاختياري **F** فهو يمثل اسماً لدالة موجودة في الصفحة تُستدعى لحظة انتهاء تنفيذ الطلب و يمكن تمرير استجابة الخادم كوسيط لهذه الدالة.

و هنا يجدر التنبيه أن هذه الدالّة تعتبر من دوال **jQuery** الوظيفيّة (مجموعة من دوال المكتبة تُستدعى مباشرةً دون الحاجة للمحددات و يكون لهذه الدوال الصيغة **$.X** حيث أن **X** هنا يمثل اسم الدالة) .

و كمثال على استخدام الدالة الوظيفية **$.get** قد نكتب مايلي :

```
$.get(
  '/myPage.php',
  {p1 :'10' , p2 :'cat'},
  Function(data){alert(data);}
);
```

حيث أن تنفيذ الشيفرة السابقة يرسل طلباً غير متزامن إلى الصفحة :

/myPage.php?p1=10&p2=cat

و يأخذ القيمة التي تعيدها هذه الصفحة ليسندها للكائن **data** الذي يمثل وسيطاً لدالة بسيطة تقوم بعرض قيمة هذا الوسيط في صندوق رسالة .

إرسال طلبات من النوع POST :

و في سياق مشابه للدالة الوظيفية **$.get** تقوم أختها الدالة الوظيفية **$.post** بإرسال طلب غير متزامن من النوع POST إلى الخادم و شكلها التالي مشابه لشكل أختها :

```
$.post(url,JSON,F);
```

حيث أن الوسيط **url** يمثل اسم الصفحة في الخادم ، و JSON يمثل وسطاء هذه الصفحة ، و **F** يمثل اسماً لدالة تستدعى عند انتهاء تنفيذ الطلب و يمكن أن تمرر استجابة الخادم كوسيط لها .

إرسال الطلبات لخادم معلوم نوع الاستجابة :

من الممكن أن يعيد التطبيق الموجود في الخادم استجابته في أي شكل يحدده مبرمج هذا التطبيق ، فإن كنا على بينة و يقين أن التطبيق الذي نرسل إليه الطلب سيعيد استجابته بالتمثيل JSON فإن المكتبة **jQuery** توفر دالةً خاصة لمثل هذه الحالة تشابه كثيراً الدالة الوظيفية **$.get** عدا أنها مصممة لاستعادة كائنات JSON ، هذه الدالة تحمل الاسم **$.getJSON** و لها الشكل التالي :

```
$.getJSON(url,JSON,F);
```

حيث أن الوسيط **url** يمثل الصفحة الهدف في الخادم و JSON يمثل وسطاء هذه الصفحة ، أما الوسيط **F** فيمثل اسماً لدالة في المستند تستدعى لحظة انتهاء الطلب و يمكن أن يمرر لها وسيط يعبر عن استجابة الخادم و يكون هذا الوسيط كائناً من كائنات JSON.

و ما دمنا في سياق الحديث عن الدوال الوظيفية الخاصة بـ **AJAX** في مكتبة **jQuery** فلا ضير أن نمر على الدالة **$.getScript** التي تقوم بجلب ملف **java script** من الخادم و تنفيذه داخل الصفحة بشكل غير متزامن و هذه الدالة لها الشكل :

```
$.getScript(url,F);
```

حيث أن الوسيط **url** يمثل ملف الـ**Java script** في الخادم و يمثل الوسيط **F** اسماً لدالة موجودة في المستند تستدعى لحظة اكتمال الطلب .

التحكّم الكامل بطلبات AJAX :

حسناً .. كما أن المكتبة **jQuery** تمنحنا دوالاً بسيطةً لإنشاء طلبات **AJAX** غير متزامنة فإنها تمنحنا دالةً تتيح لنا إنشاء طلبات **AJAX** مع التحكم الكامل بكل ما يتعلق بكل من هذه الطلبات ، هذه الدالة هي الدالة الوظيفية **$.ajax** و التي لها الشكل العام التالي :

```
$.ajax(options);
```

حيث أن الوسيط **options** هو عبارة عن كائن **JSON** يمثل مجموعة الخيارات الخاصة بالطلب و التي سيتم تنفيذه بناءً عليها . يوضح الجدول التالي خيارات الدالة الوظيفية **$.ajax**:

الاسم	النوع	الوصف
url	نص	رابط الصفحة التي سيرسل إليها الطلب في الخادم
type	نص	طريقة إرسال الطلب ، GET أو POST

data	كائن JSON	كائن JSON يمثل مجموعة وسطاء الصفحة التي سيرسل إليها الطلب
dataType	نص	قيمة من القيم التالية التي تعبر عن نوع البيانات التي نتوقع أن يعيدها الخادم و الأنواع هي : XML و JSON و SCRIPT و TEXT
timeout	رقم	يمثل الزمن الأعظمي لتنفيذ الطلب بالمللي ثانية فإن لم يتم اكتمال الطلب خلال هذا الزمن فإن الدالة تقوم بإلغاء الطلب معتبرة حدوث خطأ ما
success	دالة	دالة يتم استدعاؤها لحظة اكتمال الطلب بنجاح
error	دالة	دالة يتم استدعاؤها لحظة حصول خطأ ما في الطلب
complete	دالة	دالة يتم استدعاؤها لحظة اكتمال الطلب سواءً أتم بنجاح أم لا
beforeSend	دالة	دالة يتم استدعاؤها قبل إرسال الطلب تماماً
async	قيمة منطقية	عندما تسند إليها القيمة True يتم إرسال الطلب بشكل غير متزامن و هي

		الحالة الافتراضية و العكس في حال إسناد القيمة **False**
processData	قيمة منطقية	بشكل افتراضي يتم ترميز البيانات المرسلة بواسطة الوسيط **data** إلى ترميز ملائم لمستعرض الـ **Web** و هي الحالة ذاتها التي تحدث عند إسناد القيمة **true** لهذه الخاصية و العكس عند إسناد القيمة **false**

و كمثال على استخدام هذه الدالة قد نكتب شيئاً مماثلاً لما يلي :

```
$.ajax(
 {
   url :'/myPage.jsp' ,
   type :'GET' ,
   dataType : 'XML'
 }
);
```

و من الجدير بالذكر أن المكتبة jQuery تسمح لنا بتعيين إعدادات افتراضية لكل الطلبات المرسلة عبر الدالة $.ajax عن الطريق الدالة الوظيفية $.ajaxSetup التي تمتلك الصيغة التالية :

```
$.ajaxSetup(properties);
```

حيث أن الوسيط `properties` هو كائن JSON يمثل مجموعة الخاصيات التي ستصبح افتراضيةً للدالة `$.ajax` و يمكن أن يحتوي الوسيط `properties` الخيارات ذاتها الموضحة في الجدول السابق ، و كمثال على عمله يمكن أن نكتب شيئاً مشابهاً لمايلي لتغيير الإعدادات الافتراضية الخاصة بالدالة `$.ajax` :

```
$.ajaxSetup(
 {
   type :'GET' ,
   dataType :'XML' ,
   error :function(err){
     alert('erro message is :' +msg);
   },
   timeout :10000
 }
);
```

الأحداث الخاصّة بطلبات AJAX :

آخر ما يجب ذكره في سياق الحديث عن استثمار مكتبة `jQuery` لكتابة شيفرات AJAX هو أن المكتبة تمتلك مجموعة من الأحداث الخاصة بطلبات AJAX و التي تستثمر تماماً

كاستثمار الأحداث الأخرى التي كنا قد مررنا عليها في دقائق الأحداث Events و الجدول التالي يوضح هذه الأحداث :

اسم الحدث	لحظة التفجير
ajaxComplete	عند انتهاء أي طلب من طلبات AJAX
ajaxError	عند فشل أي طلب من طلبات AJAX
ajaxSend	قبل إرسال أي طلب من طلبات AJAX
ajaxStart	عند بداية إرسال أي طلب من طلبات AJAX
ajaxStop	عند انتهاء تنفيذ جميع طلبات AJAX
ajaxSuccess	عندا انتهاء تنفيذ أي من طلبات AJAX بنجاح

الآن أستطيع أن أقول أننا انتهينا من تعلم كيفية استثمار مكتبة الـ Web الرائعة jQuery لتنفيذ طلبات AJAX و هكذا نصل إلى نهاية الدقيقة 110 و بما أننا اتفقنا على 120 دقيقة في بداية الكتاب سنخصّص الدقائق المتبقية لاستعراض أهم إضافات المكتبة و تعلم كيفية بناء إضافات خاصة بنا .

من الدقيقة 110 إلى الدقيقة 120 :

الإضافات

Plugins

الإضافات plugins :

يسمح لنا فريق تطوير المكتبة jQuery بمشاركته في الإبداع و الابتكار بفتح باب توسعة المكتبة عبر ما يعرف بالإضافات plugins إذ يمكننا إنشاء إضافة خاصّة بنا توسِّع عمل المكتبة في أيٍّ من نواحي عملها و من ثم مشاركة هذه الإضافة مع جميع مستخدمي المكتبة عبر تبويب الإضافات في موقعها الإلكتروني على العنوان plugins.jquery.com .

عند الحديث عن الإضافات هناك طريقان يسلك المتحدث أحدهما ، الطريق الأول : الحديث عن كيفية كتابة الإضافات الخاصّة بالمكتبة و هنا نضطر للعودة إلى لغة Java Script التقليدية ، أما الطريق الثاني : الحديث عن استخدام إضافات jQuery الموجودة أصلاً و هذا هو الطريق الذي سنسلكه في هذه الدقائق أولاً .

في الحقيقة ليس هنالك طريقة ثابتة لاستخدام جميع إضافات المكتبة فالطريقة تختلف من إضافة إلى أخرى و لكن بشكل عام يمكننا القول :

لاستخدام إضافة من إضافات jQuery الموجودة يكفي أن تقوم بتحميل هذه الإضافة التي تأتي على شكل ملف Java Script ثم تقوم بتضمينه ضمن صفحة الـ Web الخاصة بك بعد تضمين المكتبة و هذه نقطة هامة إذ أنَّ الإضافات تُبنى بالاعتماد على المكتبة و بالتالي عليكَ التأكُّد من تضمين المكتبة أولاً ، بعد تضمين الإضافة سيصبح هناك دوال جديدة تضاف إلى دوال المكتبة الأصلية و توسّع عملها و يمكنك استخدامها بنفس طريقة استخدام دوال المكتبة التي شرحناها في دقائق الحديث عن الدوال .

حسناً .. هذا الكلام كلام عام لا يغني عن قراءة التوثيق الخاص بكل إضافة و الذي لا يتجاوز الصفحة عادةً ، و الآن يمكننا البدء في استعراض بعض إضافات المكتبة على سبيل المثال لا على سبيل الحصر و التمييز .

الإضافة jQuery user Interface :



تعتبر هذه الإضافة الأكثر شعبيةً بين جميع إضافات المكتبة على الإطلاق إذ تعد بتقديم واجهات استخدام أنيقة و غنيّة لصفحتك (بعضها كَتلكَ التي رأيناها في موقع الشّركة العملاقة Microsoft) و هذا ما يعنيه اسم الإضافة ، و تقدّم الإضافة مجموعة من الأدوات الجاهزة القابلة لإعادة الاستخدام و التي تمنح واجهة التطبيق الخاص بال Web مرونةً عاليةً لدرجة تجعله مشابهاً تقريباً لتطبيق سطح المكتب من ناحية واجهة الاستخدام ، فهي تقدم الألسنة tabs و أشرطة التمرير scroll bars و الأوكرديونات accordions و المزاليج sliders و أدوات اختيار التاريخ date pickers و أدوات اختيار الألوان color pickers و الكثير غيرها ، كما تضيف الأداة قدراتٍ خاصّة لعناصر الصفحة لا تتوفّر فيها بشكل طبيعي مثل القدرة على سحبها و إفلاتها أو إعادة ترتيبها أو تغيير حجمها أو الاختيار منها و الكثير الكثير و يجدر الإشارة أن هذه الإضافة قابلة للإكساء و هي تمتلك عدة إكساءات جاهزة Themes بشكل افتراضي .

يمكن الحصول على هذه الإضافة المجّانيّة عن طريق الدخول إلى موقعها **www.jqueryui.com** ثمّ النقر على الزر **Build Custom Download**.

الإضافة jquery Form:

تعطي هذه الإضافة مزيداً من المرونة عند التعامل مع النماذج **Forms** إذ أنها "تكمل النقص الموجود في دوال مكتبة **jQuery** الخاصة بالتعامل مع النماذج" على حد قول مبدعيها ، و تنقسم دوال هذه الإضافة إلى عدة مجموعات منها ما يسمح باستعادة القيم الخاصة بأدوات النماذج بطريقة أبسط من طرق المكتبة و يظهر ذلك جلياً عند الحديث عن أداة نموذج مثل الأداة **<select>** ، و من هذه المجموعات ما يسمح بإعادة تعيين (تفريغ) قيم أدوات النموذج ، و منها ما يسمح بإرسال قيم النموذج إلى الخادم بطريقة تقليدية و باستخدام **AJAX** ، يمكن تحميل الإضافة من الموقع :

http://www.jquery.com/plugins/project/form

الإضافة jmaxinput:



هل سبق و قمت بزيارة الموقع الاجتماعي الشّهير **Twitter** ؟ هل لفتت طريقة عرض مربعات إدخال النصوص في ذلك الموقع انتباهك ؟

إن كنت من المهتمين بكيفية بناء مربّعات النصوص تلكَ فإنّ هذه الإضافة تجعل العملية أسهل من السُّهولة و تجعلكَ تتبع الطّريقة التي علّمتنا إيّاها **jQuery** ألا وهي طريقة "إلقاء التعقيدات وراء ظهورنا" على حد قولي ، يمكن تحميل الإضافة من الرابط :

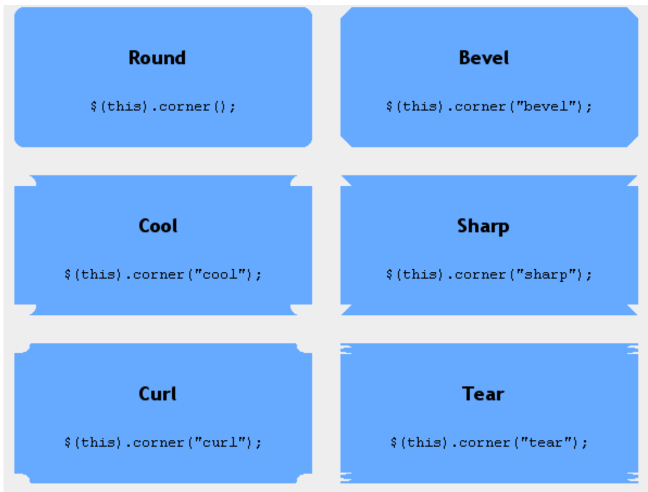
http://plugins.jquery.com/project/jMaxInput

الإضافة jquery short access:

تسمح هذه الإضافة الرائعة بتعيين مفاتيح اختصار للروابط الموجودة بالصفحة إذ يمكن مثلاً أن يكون المفتاح e مفتاح اختصار للرابط الذي يقود لموقع مايكروسوفت مثلاً ، و يكون المفتاح x مفتاح اختصار للرابط الذي يقود لموقع مكتبة jQuery و هكذا ، الجميل في هذه الإضافة أنّها تتفهم متطلبات AJAX جيّداً فلا تفسد عملها ! ، يمكن تحميل هذه الإضافة من الرابط :

http://plugins.jquery.com/project/shortaccesskey

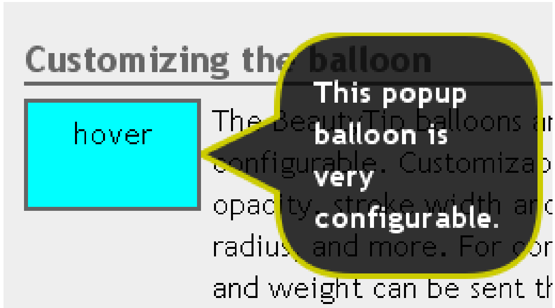
الإضافة jquery corner:



أصبح من الشائع في جميع مواقع الـ Web جيّدة التصميم ألّا تكونَ أركان (زوايا) أقسام الموقع بشكل ركن المربّع التقليدي و إنّما بأشكال مختلفة لعلَّ أشهرها الشكل الدائري Round Corner ، يمكن إنشاء هذه الأركان بطرق مختلفة أسهلها استخدام هذه الإضافة التي يمكن تحميلها من الرابط :

http://www.malsup.com/jquery/corner/

الإضافة beauty tips:



تسمح هذه الإضافة بإظهار شرح مختصر tooltip جميل جداً عند وضع المؤشر على عنصر معين من عناصر الصفحة و يمكن إظهار هذا الشرح بالشكل الذي ترغب به تماماً إذ أن الإضافة تعطي لمستخدمها خيارات كثيرة لتخصيص شكل الشرح المختصر بما يتلاءم و موقعه ، يمكن تحميل هذه الإضافة من الرابط :

http://www.lullabot.com/files/bt/btlatest/DEMO/index.html

الإضافة BOXY :

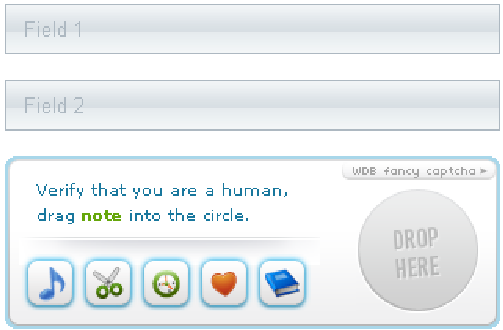


تسمح هذه الإضافة بإنشاء مربعات حوار منبثقة شاع انتشارها كثيراً في المواقع الحديثة في الفترة الأخيرة ، و هي مشابهة لتلك التي نراها في أغلب المواقع العالمية كـ Facebook على سبيل المثال بطريقة بسيطة .

يمكن تحميل هذه الإضافة من الرابط :

http://plugins.jquery.com/project/boxy

الإضافة ajax fancy captcha :

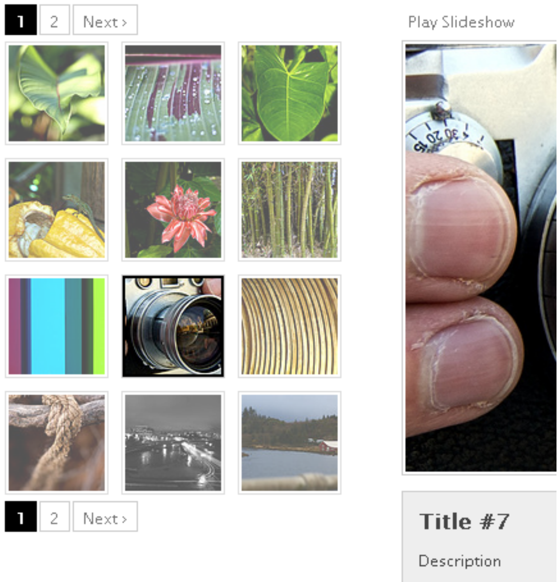


في كثير من المواقع الحديثة يلجأ المبرمجون إلى استخدام "كود التأكيد" لتفادي ما يعرف بال Bots ، تؤمّن jQuery إضافةً تسمح لنا بإضافة هذه الإمكانية في موقعنا بشكل عصري أجمل من الشكل التقليدي "ظهور صورة غير مفهومة مثلاً" فهي تطلب سحب عنصر معيّن و إفلاته في المنطقة المخصصة و بالطبع هذا العنصر يختلف من دخول إلى آخر فقد يطلب مثلاً سحب شكل المقص و إفلاته في المنطقة المخصصة في الدخول الأوَّل و قد يطلب منك سحب شكل الساعة و إفلاته في المنطقة المخصصة في الدخول الثاني و هكذا .. فإنْ كان زائر موقعنا من الـ Bots فإنّه لن يتجاوز هذا الاختبار البسيط جداً بالنسبة لابنك أو لأخيك الصغير.

يمكن تحميل هذه الإضافة من الرابط :

http://www.webdesignbeach.com/beachbar/ajax-fancy-captcha-jquery-plugin

الإضافة Gallerific:



تسمح لنا هذه الإضافة بإنشاء معرض صور بالميزات الأساسية التي تتوفر في معارض الصور على الانترنت ببساطة شديدة ، يمكن تحميلها من الرابط :

http://www.twospy.com/galleriffic/

المزيد من الإضافات ؟

ما عرضناه في الدقائق القليلة الماضية يعتبر أنموذجاً فقط من الإضافات الكثيرة جداً الخاصة بالمكتبة و التي يمكن أن نطلع على آخر ما يضاف إليها عبر ركن الإضافات في موقع المكتبة على العنوان :

http://plugins.jquery.com

و الآن و بعد أن رأينا جمال إضافات المكتبة حان الوقت لنتعلّم كيفية إنشاء هذه الإضافات.

كيفية إنشاء الإضافات plugins :

كما قلنا سابقاً فإن الإضافات توسّع عمل المكتبة في أيٍّ من نواحي عملها و يعطينا فريق تطوير المكتبة الحرية الكاملة في توسعة عمل مكتبته شرط اتباع مجموعة من القواعد التي نناقشها في هذه الدقائق .

تسمية ملفّات الإضافات :

كنت قد ذكرت في مقدمة هذه الدقائق أن الإضافة ما هي إلا ملف **Java Script** يتم تضمينه بعد المكتبة و يبنى بالاعتماد عليها و بما أنَّ مجال كتابة الإضافات متاح للجميع فإنَّ فريق تطوير المكتبة ينصح بتبنّي طريقة التسمية التالية للملفات و هي ليست قاعدةً و إنما نصيحةً أصبحت عُرفاً فيما بعد :

"سمِّ ملف الإضافة الخاصّة بك باسمٍ من الشكل التالي : jQuery.plugin.js"

حيث أن **plugin** هو اسم إضافتك ، فمثلاً لو قمت بإنشاء إضافة تحمل اسمي يجب أن أسمِّي الملف الذي يحتوي إضافتي بالاسم **jQuery.Mukhtar.js** إنْ كنتُ ممَّن يستمعون قولَ فريق تطوير المكتبة و يتبّعون أأَمَنه .

الشكل العام لشيفرة الإضافة :

حسناً .. بعد أن أنشأنا الملف الخاص بإضافتنا بالاسم الذي نصحَنا به فريق تطوير المكتبة سنكتب فيه شيفرة برمجية لها الشكل العام التالي :

```
(function($){

        شيفرة إضافتنا ستكون هنا

})(jQuery);
```

بهذه الطريقة نضمن أن الدوال التي سنقوم بتعريفها في المنطقة المخصصة لنا ستضاف إلى المكتبة و نضمن أننا سنستطيع استدعاءها بطريقة استدعاء دوال المكتبة الأصلية .

طريقة بناء دوال الإضافات :

إذا كنت تذكر حديثنا في الدقائق الماضية أجمعِها منذ بداية الكتاب فمن المفترض أنك الآن تعرف تماماً أنَّ المكتبة تمتلك نوعين من الدوال : نوع أول يطبق على مجموعة من عناصر المستند بالتعاون مع المحدِّدات و نوع ثانٍ أطلقنا عليه لقب "الوظيفي" يُستدعى دون مساعدة المحدِّدات ، حسناً تختلف طريقة بناء دوال الإضافات باختلاف هذين النوعين فلبناء دالة جديدة من النوع الأول الذي يطبق على مجموعة من عناصر المستند بالتعاون مع المحدِّدات فإننا نتبع الشكل العام التالي :

```
(function($){

      $.F = function(Parameters)

      {

         جسم الدالة هنا

      };

})(jQuery);
```

حيث أنَّ **F** هو اسم الدالة الجديدة و **Parameters** هي قائمة الوسطاء التي تأخذها هذه الدالة.

أمّا عند بناء دالة وظيفية جديدة من النوع الثاني الذي تطبّق دوالّه دونَ استخدام المحدِّدات نتبع الشكل العام التالي :

```
(function($){
```

```
    $.fn.F = function(Parameters)
    {
        جسم الدالة هنا
    };
})(jQuery);
```

حيث أنَّ F هو اسم الدالة الجديدة و Parameters هي قائمة الوسطاء التي تأخذها هذه الدالة.

أمثلة على إنشاء إضافات خاصَّة :

حسناً .. الأفكار السابقة هي الأفكار الرئيسية لبناء الإضافات و كمثال على بناء دالة وظيفية جديدة بسيطة جداً يمكن أن نكتب مايلي :

```
(function($){
    $.ourMessage = function(msg)
    {
      var m=msg.toString();
       alert(m);
    };
})(jQuery);
```

و كما هو واضح تقوم هذه الدالة بعرض رسالة معلومات تحتوي على قيمة الوسيط الممرر إليها و الآن يمكننا أن نجرب إضافتنا بكتابة شيفرة jQuery كمايلي :

```
$.ourMessage('Hello world ! ');
```

و كمثال على دوال النوع الآخر التي تطبّق على عناصر المستند التي يعيدها المحدِّد سنقوم بكتابة دالّة جديدة تغيّر نوع الخط الخاص بجميع العناصر التي يعيدها المحدد إلى النوع Arial و ستكون شيفرة هذه الدالة كمايلي :

```
(function($){

      $.fn.changeFont=function()

      {

        return this.css('font-family', 'Arial');

      };

})(jQuery);
```

لاحظ أننا استعملنا الدّالة css إحدى دوال jQuery التي كنا قد ناقشناها سابقاً في دقائق الحديث عن الدوال و ما يجب ذكره هنا هو استعمالنا للكلمة return قبل استدعاء الدالة لكي نسمح لدالتنا أن تُستدعى مع الدوال الأخرى المعتمِدة على المحدِّدات فلو أردنا اختبار دالتنا مع دالة أخرى الآن يمكننا أن نكتب ببساطة :

```
$('#Mydiv').changeFont().fadeIn();
```

مما سيؤدّي إلى تطبيق حركة الظهور المتلاشي بعد تغيير نوع الخط و يجدر بنا أن نذكر هنا أن دالتنا حالياً تستطيع التعامل مع كائن واحد فقط من كائنات المستند و لهذا استعملتُ محدِّداً يعيد كائناً واحداً في المثال (أعني المحدد #Mydiv) .

و بالطبع يمكن تحسين الدالة بمثل كل الحالات المشابهة السابقة بالاعتماد على الدوران each ليصبح شكل دالتنا النهائي هو :

```
(function($){
  $.fn.changeFont = function(){
    return this.each(function() {
      $(this).css('font-family' , 'arial');
    });
  };
})(jQuery);
```

و هكذا تصبح دالتانا السابقتان جاهزتين للنشر في إضافة لن يستخدمها أحد لأنها إضافة لا تقوم بأي شيء جديد عدا تعليمنا كيفية بناء الإضافات ☺ .

الخاتمة :

و الآن و قد شارفنا على الدّخول في الدقيقة 121 يجدر بي أن أطفئ الأنوار و أنسحب بهدوء و لكن بعد الاعتراف بأنني لا أزال أتعلَّم و أرحِّب بأي استفسار و بكل من يريد أن يتواصل معي عبر البريد الإلكتروني mokhtar_ss@hotmail.com

برمجةً ممتعة ... و إلى لقاء قادم في دقائق أخرى بإذن الله.

تمَّ بحمد الله عزَّ و جَل

دمشق 8/1/2010

جدول المحتويات

أساسيَّات JQUERY.......... **5**

تثبيت jQuery و تضمينها في صفحتك 6

أساسيات jQuery 7

المزيد من الأمثلة ؟ 10

طريقة استعمال jQuery 10

ما معنى الشيفرة السابقة ؟ 12

المُحَدِّدات SELECTORS.......... **13**

المُحَدِّدات 14

توليد محتويات HTML جديدة 30

الدَّوال FUNCTIONS.......... **32**

الدَّوال Functions 33

دوال التعامل مع واصفات الوسوم Attributes 33

دوال التعامل مع الأنماط Styles 37

دوال التعامل مع محتوى عناصر المستند Inner content 44

دوال التغليف Wrapping 51

دوال حذف عناصر المستند 54

دوال التعامل مع عناصر النماذج FORM ELEMENTS 54

الأحداث EVENTS 58

الحَركات ANIMATIONS 72

الحَركات Animations 73

المؤثرات البسيطة 73

مؤثرات التلاشي 79

مؤثّرات الانزلاق 81

إيقاف الحركة 82

مرجع سريع لحركات المكتبة 82

إنشاء حركات خاصة 83

التخاطب مع الخادم عبر تقنيّة AJAX 86

جلب المحتوى من الخادم 87

تمرير وسطاء للخادم من حقول النماذج 89

إرسال طلبات من النوع GET 89

إرسال طلبات من النوع POST 91

إرسال الطلبات لخادم معلوم نوع الاستجابة 91

التحكّم الكامل بطلبات AJAX 92

الأحداث الخاصّة بطلبات AJAX 95

الإضافات PLUGINS 97

الإضافات plugins 98
الإضافة jQuery user Interface 99
الإضافة jQuery Form 100
الإضافة jmaxinput 100
الإضافة jQuery short access 101
الإضافة jQuery corner 101
الإضافة beauty tips 102
الإضافة BOXY 102
الإضافة AJAX fancy captcha 103
الإضافة Gallerific 104
المزيد من الإضافات ؟ 104
كيفية إنشاء الإضافات plugins 105
تسمية ملفّات الإضافات 105
الشكل العام لشيفرة الإضافة 105
طريقة بناء دوال الإضافات 106
أمثلة على إنشاء إضافات خاصّة 107
الخاتمة 107

المؤلف في سطور

- مختار فؤاد سيِّد صالح.
- نشرت له أعمال شعريَّة باسم مستعار هو (مختار الكمالي) نسبةً إلى مسقط رأسه.
- من مواليد مدينة البوكمال – سوريا – 1989م.
- يحمل إجازة في هندسة الحاسوب و المعلوماتيَّة – الجامعة السوريّة الدوليّة – 2011م.
- يدرس حالياً ماجستير Web Technologies.
- حصل على جوائز عديدة في مجال البرمجيَّات و في مجال الشِّعر.
- صدر له:
  - إصدارات علميّة:
    - تعلَّم HTML5 & CSS3 – تحت الطبع.
    - تعلَّم jQuery في 120 دقيقة – إصدار خاص – دمشق – 2010م.
    - 3D Game Studio طريقك نحو برمجة الألعاب – كتاب إلكتروني ملحق مع مجلَّة F1-Magazine التقنية السورية – 2006م.
  - إصدارات أدبيَّة:
    - في غيابة الحُبِّ – شعر – دائرة الثقافة و الإعلام بالشارقة 2012م.
- للتواصل: facebook.com/mukhtar.ss